O NOUA ABORDARE A PROBLEMATICII INFASURARII SUPRAFETELOR, BAZATA PE METODE DE REPREZENTARE IN FORMA DISCRETA A SUPRAFETELOR, IN VEDEREA ALGORITMIZARII SI INFORMATIZARII PROFILARII SCULELOR GENERATOARE

Raport anual grant ID_656
Contract 238/2007

Colectiv: prof. dr. ing. Nicolae OANCEA
prof. dr. ing. Vasile MARINESCU
s.l. dr. ing. Virgil Gabriel TEODOR
cercet. dr. ing. Marian CUCU
cercet. drd. ing. Ionut POPA
cercet. drd. ing. Gabriel DURA

Anul 2010

CUPRINS

OBIECTIVUL 9. SINTEZA UNOR MODALITĂȚI DE REPREZENTARE PRIN POLI A MATRICELOR DE COORDONATE REPREZENTÂND SUPRAFEȚE EXPRIMATE ÎN FORMĂ DISCRETĂ

9.1. Forme de reprezentare prin poli a suprafeței elicoidale cilindrice

Introducere

Suprafețele elicoidale cilindrice şi de pas constant cunoscute printr-o matrice de coordonate obținută prin măsurare directă pe maşini de măsurat în coordonate 3D, vezi figura 9. 1, în forma unei matrice,

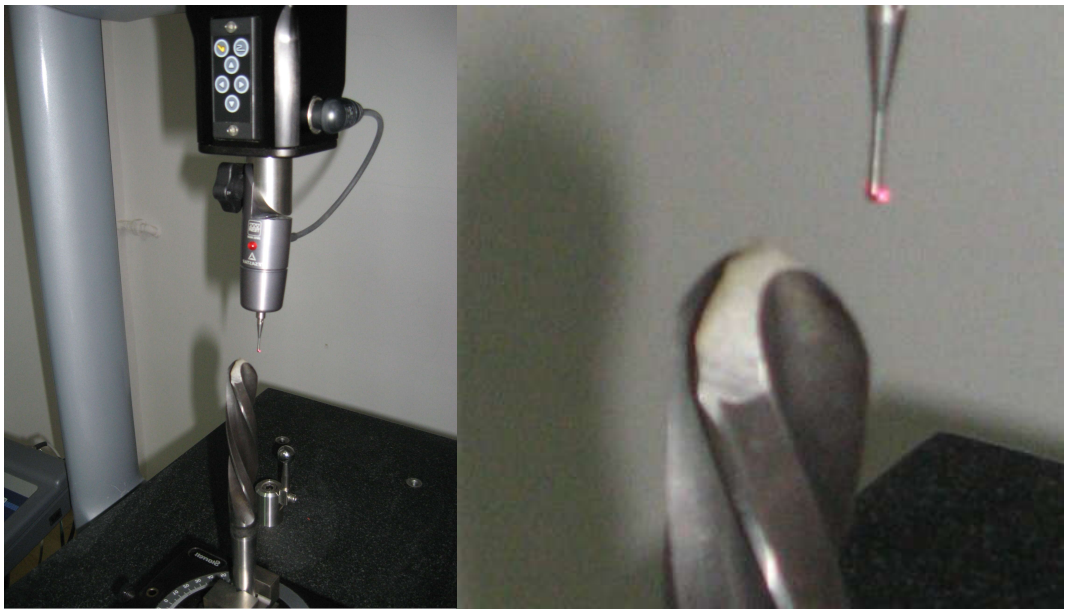


Figura 9. 1. Măsurarea de coordonate pe generatoarea unei suprafețe elicoidale cilindrice de pas constant

$$G = \begin{Vmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ \dots & \dots \\ X_n & Y_n \end{Vmatrix} \tag{9.1}$$

conduc la o reprezentare în planul XY de forma, vezi figura 9. 2:

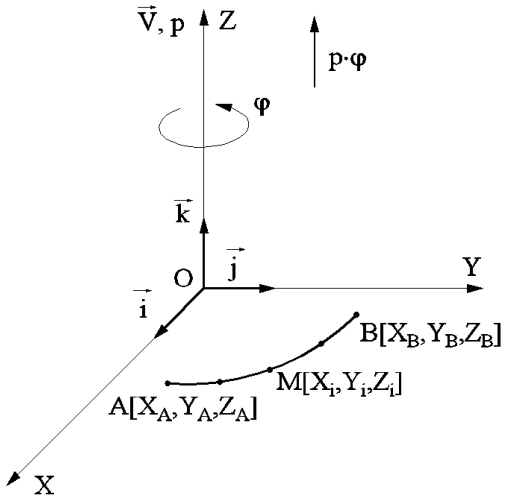


Figura 9. 2. Generatoarea discretă a suprafeței elicoidale

Se substituie generatoarea discretă G (9.1) cu un polinom Bezier de grad inferior, care să permită o reprezentare cât mai riguroasă a ansamblului de coordonate, asiguîndu-se:

- Coeficientul de determinare,R^2 ,a polinomului de substituție față de coordonatele efectiv măsurate (cât mai aproape de valoarea 1).
- derivata a doua a polinomului să fie liniară, pe tot intervalul de definiție.

Pentru un polinom de gradul II, se obține forma:

$$\begin{aligned} X &= A_X \cdot \lambda^2 + 2\lambda \cdot (1-\lambda) \cdot B_X + (1-\lambda)^2 \cdot C_X; \\ Y &= A_Y \cdot \lambda^2 + 2\lambda \cdot (1-\lambda) \cdot B_Y + (1-\lambda)^2 \cdot C_Y; \end{aligned} \tag{9.2}$$

$0 \le \lambda \le 1$.

Coeficienții polinomului, $A_X, A_Y, B_X, B_Y, C_X, C_Y$,se obțin din condițiile, vezi figura 9. 3:

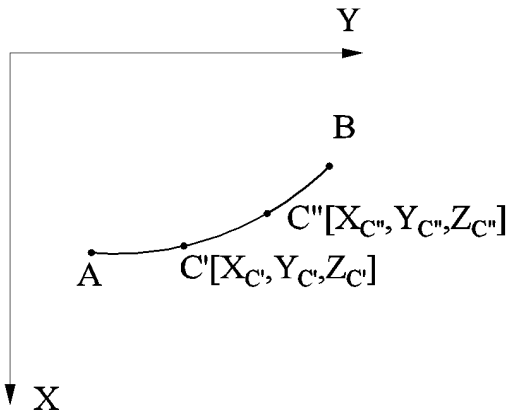


Figura 9. 3. Coeficienții polinomului Bézier

pentru λ= 0,

$$\begin{aligned} X_A &= C_X \\ Y_A &= C_Y \end{aligned} \tag{9.3}$$

pentru λ= 1,

$$\begin{aligned} X_B &= A_X \\ Y_B &= A_Y. \end{aligned} \tag{9.4}$$

Este dificil ca în urma măsurătorii să se definească un punct C' astfel încât să existe egalitate între segmentele

$$\overline{AC'} = \overline{C'B}, \tag{9.5}$$

situație care să corespundă unei valori pentru λ =0.5.

Atunci, se acceptă că se pot măsura două puncte, C', respectiv C", care se află în apropierea mijlocului arcului $\overset{\frown}{AB}$, (deci, pentru λ= 0.5) şi se calculează o medie a valorii lui λ, pentru punctele efectiv măsurate astfel:

$$\lambda_{C'} = \frac{\overline{AC'}}{\overline{AC'} + \overline{C'B}} \tag{9.6}$$

$$AC' = \sqrt{(X_A - X_{C'})^2 + (Y_A - Y_{C'})^2} \tag{9.7}$$

$$C'B = \sqrt{(X_{C'} - X_B)^2 + (Y_{C'} - Y_B)^2} \tag{9.8}$$

şi

$$\lambda_{C''} = \frac{\overline{AC''}}{\overline{AC''} + \overline{C''B}} \tag{9.9}$$

cu definițiile

$$AC'' = \sqrt{(X_A - X_{C''})^2 + (Y_A - Y_{C''})^2}\ ; C''B = \sqrt{(X_B - X_{C''})^2 + (Y_B - Y_{C''})^2}\ . \tag{9.10}$$

Se acceptă valoarea parametrului λ ca fiind

$$\lambda_{central} = \frac{\lambda_{C'} + \lambda_{C''}}{2}\ . \tag{9.11}$$

Este evident, $\lambda_{central}$ (λ_C) nu are valoarea 0.5 dar eroarea de aproximare este satisfăcătoare din punct de vedere tehnic.

Se determină coeficienții B_X şi B_Y în baza lui $\lambda_{central}$ (λ_C).

Pentru $\lambda_{C'}$, calculat anterior, rezultă:

$$\begin{aligned} X_{C'} &= A_X \cdot \lambda_{C'}^2 + 2\lambda_{C'} \cdot (1-\lambda_{C'}) \cdot B_X + (1-\lambda_{C'})^2 \cdot C_X; \\ Y_{C'} &= A_Y \cdot \lambda_{C'}^2 + 2\lambda_{C'} \cdot (1-\lambda_{C'}) \cdot B_Y + (1-\lambda_{C'})^2 \cdot C_Y, \end{aligned} \tag{9.12}$$

ecuații care permit determinarea unei prime mărimi a constantelor B_X şi B_Y, pe care le definim $B_{XC'}$ şi $B_{YC'}$.

În mod similar, se determină constantele polinomului care corespund punctului C", notate $B_{XC''}$, $B_{YC''}$.

Astfel, polinoamele de aproximare a coordonatelor măsurate vor fi definite pentru constantele:

$$B_X = \frac{B_{XC'} + B_{XC''}}{2};\ B_Y = \frac{B_{YC'} + B_{YC''}}{2}. \tag{9.13}$$

Sunt definite, vezi (9.3), (9.4) şi (9.13) constantele polinomului de substituire a punctelor măsurate ale generatoarei (9.1).

Suprafața elicoidală cilindrică şi de pas constant, având ca generatoare matricea de coordonate (9.1),se determină în mişcarea elicoidală

$$X = \omega_3^T \cdot (k \cdot \Delta\theta) \cdot \begin{Vmatrix} X(\lambda) \\ Y(\lambda) \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ p \cdot k \cdot \Delta\theta \end{Vmatrix} \tag{9.14}$$

în care: Δθ este incrementul unghiular de rotație în jurul axei suprafeței elicoidale;

p – parametrul elicoidal al suprafeței elicoidale;

k – număr natural.

Parametrul elicoidal al suprafeței poate fi determinat prin măsurarea suprafeței elicoidale, măsurând diametrul exterior al acesteia (D_{ex}) şi unghiul de înclinare al elicei suprafeței de pe această suprafață cilindrică (β_{ex}) figura 9. 4.

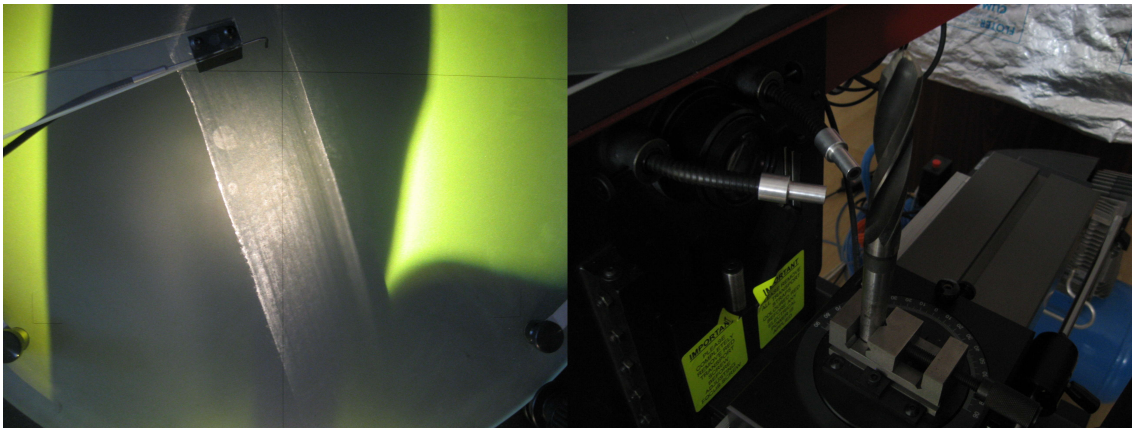


Figura 9. 4. Măsurarea unghiului de înclinare a elicei pe diametrul exterior pe profil proiector Sttarret

Din figura 9. 5, se deduce o schemă de măsurare a parametrului elicoidal al unei suprafețe date.

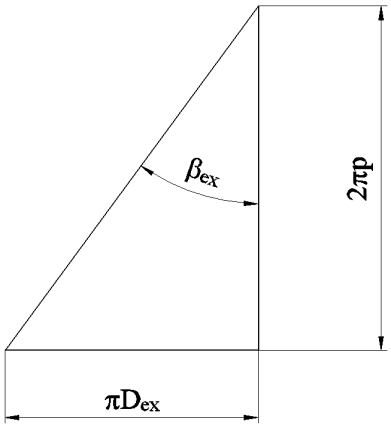


Figura 9. 5. Corelația parametrului β_{ex}.

Din figura 9. 5, rezultă

$$tg\beta_{ex} = \frac{\pi \cdot D_{ex}}{2\pi \cdot p}, \tag{9.15}$$

de unde, deoarece β_{ex} şi D_{ex} sunt măsurabile, se defineşte mărimea parametrului elicoidal al suprafeței,

$$p = \frac{D_{ex}}{2 \cdot tg\beta_{ex}} [\text{mm}]. \tag{9.16}$$

Generarea suprafeței presupune alegerea arbitrară a mărimii incrementului unghiular, $\Delta\theta$, în concordanță cu precizia de reprezentare în formă discretă a suprafeței, şi astfel, din (9.14),se pot calcula coordonatele succesive ale suprafeței reprezentată în formă discretă, cu generatoare substituite de un polinom Bézier:

$$\begin{Vmatrix} X \\ Y \\ Z \end{Vmatrix} = \begin{Vmatrix} \cos(k\cdot\Delta\theta) & -\sin(k\cdot\Delta\theta) & 0 \\ \sin(k\cdot\Delta\theta) & \cos(k\cdot\Delta\theta) & 0 \\ 0 & 0 & 1 \end{Vmatrix} \cdot \begin{Vmatrix} X(\lambda) \\ Y(\lambda) \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ p\cdot k\cdot\Delta\theta \end{Vmatrix} \tag{9.17}$$

care, după dezvoltări, este adusă la forma:

$$\begin{cases} X = X(\lambda)\cdot\cos(k\cdot\Delta\theta) - Y(\lambda)\cdot\sin(k\cdot\Delta\theta); \\ Y = X(\lambda)\cdot\sin(k\cdot\Delta\theta) + Y(\lambda)\cdot\cos(k\cdot\Delta\theta); \\ Z = p\cdot k\cdot\Delta\theta. \end{cases} \tag{9.18}$$

Funcțiile X(λ), Y(λ) sunt date în (9.2), ale căror coeficienți A_X, B_X, C_X, A_Y, B_Y, C_Y sunt determinabili prin coordonatele măsurate pe generatoarea suprafeței.

Se obține, astfel, o formă de reprezentare a suprafeței, care este o formă discretă (punct cu punct) putînd constitui baza pentru profilarea sculelor mărginite de suprafețe de revoluție (scula disc şi scula cilindro-frontală pentru generarea suprafețelor elicoidale).

Se poate accepta că parametrul λ variază continuu între 0 şi 1 şi, astfel, ecuațiile (9.18) pot fi interpretate ca o formă de reprezentare analitică a suprafeței elicoidale. Cu o astfel de reprezentare, apar posibilități diverse de profilare a sculelor generatoare ale unei astfel de suprafețe elicoidale.

Se prezintă un exemplu de aplicare a metodei propuse pentru reprezentarea suprafeței elicoidale în formă discretă.

9.1.1. Forma de reprezentare a canalului elicoidal al burghiului obținut prin măsurare punct cu punct

Pe maşina de măsurat 3D, se defineşte generatoarea canalului elicoidal, măsurată punct cu punct, vezi figura 1, şi definită de ansamblul de coordonate G_{20}, pentru un burghiu cu diametrul de 20 mm, vezi tabelul 9. 1.

Tabelul 9. 1. Generatoarea transversală a burghiului-G_{20}

X	Y	Z
2.771	10.087	1
1.83	9.665	1
0.84	9.129	1
-0.37	8.306	1
-1.4	7.298	1
-3.002	5.279	1
-3.165	4.302	1
-2.849	2.886	1
-2.149	1.766	1
-0.796	0.332	1
0.355	-0.754	1

În figura 9. 6, se prezintă forma discretă a generatoarei burghiului, (ansamblul de puncte, vezi tabelul 9. 1.).

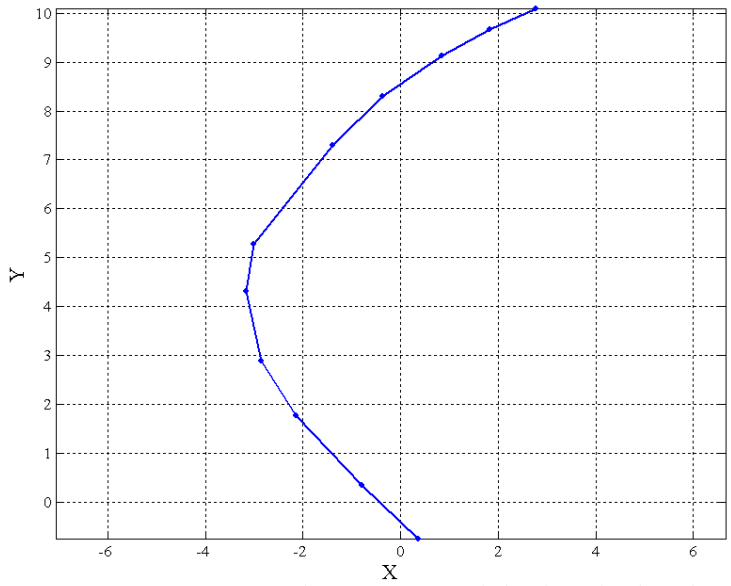


Figura 9. 6. Generatoarea discretă a canalului burghiului elicoidal

Se acceptă un polinom de substituire de gradul 3, pentru ansamblul de coordonate discrete ale generatoarei canalului elicoidal, vezi şi figura 9. 7,

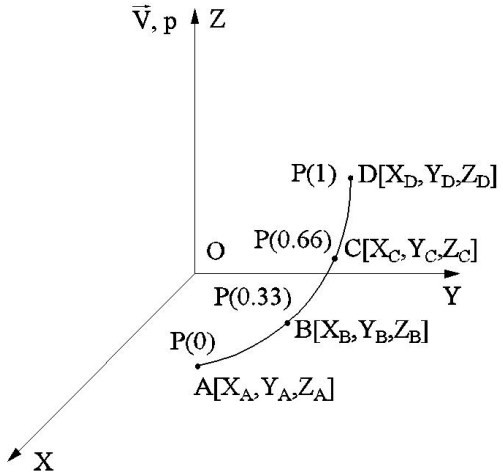


Figura 9. 7. Polinom Bézier, noduri caracteristice.

$$\begin{cases} X = A_X \cdot \lambda^3 + 3\lambda^2 \cdot (1-\lambda) \cdot B_X + 3\lambda \cdot (1-\lambda)^2 \cdot C_X + (1-\lambda)^3 \cdot D_X; \\ Y = A_Y \cdot \lambda^3 + 3\lambda^2 \cdot (1-\lambda) \cdot B_Y + 3\lambda \cdot (1-\lambda)^2 \cdot C_Y + (1-\lambda)^3 \cdot D_Y. \end{cases} \tag{9.19}$$

Se definesc mărimile parametrului λ pentru nodurile polinomului:

$$\begin{aligned} \lambda_A &= 0 \\ \lambda_D &= 1 \\ \lambda_B &= \frac{\overline{AB}}{\overline{AB} + \overline{BC} + \overline{CD}} \\ \lambda_C &= \frac{\overline{AB} + \overline{BC}}{\overline{AB} + \overline{BC} + \overline{CD}} \end{aligned} \tag{9.20}$$

în care:

$$AB = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}\ ;$$
$$BC = \sqrt{(X_B - X_C)^2 + (Y_B - Y_C)^2}\ ; \tag{9.21}$$
$$CD = \sqrt{(X_C - X_D)^2 + (Y_C - Y_D)^2}\ .$$

Ca regulă generală, λ_B şi λ_C trebuie să aibă valori cât mai apropiate de 0.33 şi, respectiv, 0.66.

Cu aceste valori ale parametrului λ, se determină coeficienţii polinomului de substituire, din sistemul de ecuaţii:

-pentru $\lambda = 0$,

$$X_A = D_X; \tag{9.22}$$
$$Y_A = D_Y;$$

-pentru $\lambda = 1$,

$$X_D = A_X; \tag{9.23}$$
$$Y_D = A_Y;$$

-pentru $\lambda = \lambda_B$,

$$X_B = A_X \cdot \lambda_B^3 + 3\lambda_B^2 \cdot (1-\lambda_B) \cdot B_X + 3\lambda_B \cdot (1-\lambda_B)^2 \cdot C_X + (1-\lambda_B)^3 \cdot D_X;$$
$$Y_B = A_Y \cdot \lambda_B^3 + 3\lambda_B^2 \cdot (1-\lambda_B) \cdot B_Y + 3\lambda_B \cdot (1-\lambda_B)^2 \cdot C_Y + (1-\lambda_B)^3 \cdot D_Y; \tag{9.24}$$

-pentru $\lambda = \lambda_C$,

$$X_C = A_X \cdot \lambda_C^3 + 3\lambda_C^2 \cdot (1-\lambda_C) \cdot B_X + 3\lambda_C \cdot (1-\lambda_C)^2 \cdot C_X + (1-\lambda_C)^3 \cdot D_X;$$
$$Y_C = A_Y \cdot \lambda_C^3 + 3\lambda_C^2 \cdot (1-\lambda_C) \cdot B_Y + 3\lambda_C \cdot (1-\lambda_C)^2 \cdot C_Y + (1-\lambda_C)^3 \cdot D_Y; \tag{9.25}$$

Ansamblul de ecuaţii (9.22)–(9.25) permite determinarea coeficienţilor polinomului de substituire.

Calitatea substituirii generatoarei efectiv măsurate (tabelul 9. 1) cu un polinom, se poate îmbunătăţi dacă în loc de soluţiile prezentate anterior, pentru punctele B şi C, se procedează aşa cum am prezentat în (9.3)–(9.13). Un număr mare de puncte măsurate (tabelul 9. 1) pot să asigure obţinerea unor valori acceptabile pentru mărimile parametrului λ, fără a mai fi necesar calculul unor valori medii pentru λ.

În figura 9. 8, este prezentată generatoarea aproximată cu elementele caracteristice ale aproximării (R^2 =0.9987) şi derivata a doua a polinomului rectilinie.

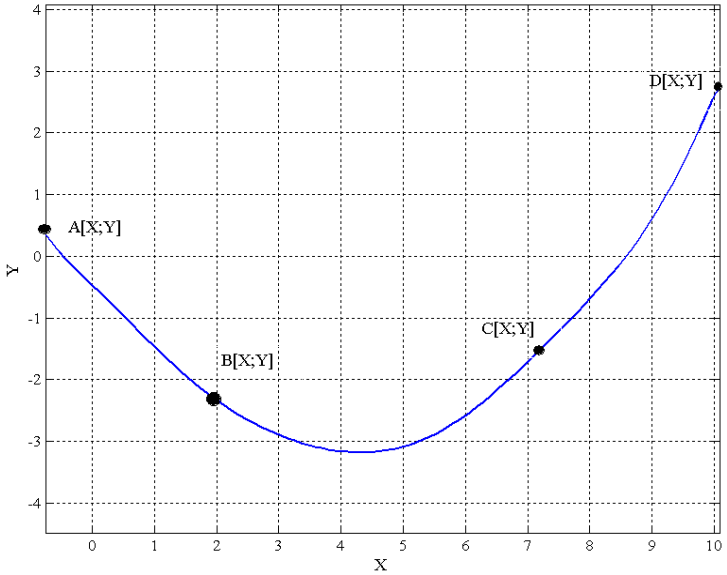


Figura 9. 8. Generatoarea aproximată a unui polinom de gradul II (secţiunea transversală a burghiului)

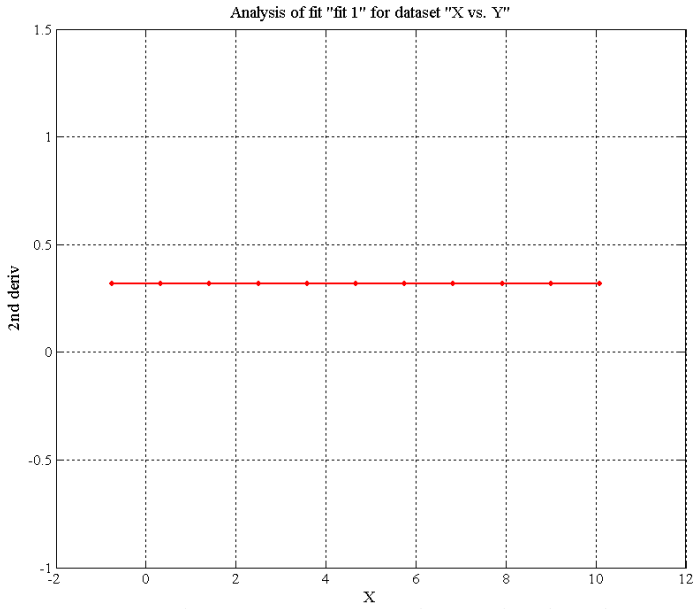


Figura 9. 9. Forma derivatei a II-a a polinomului de substituire

În figura 9. 10, în baza algoritmului prezentat, vezi (9.18), se determină forma suprafeței elicoidale a canalului sculei, reprezentată, în formă discretă, pentru un parametru elicoidal p=27.47 mm şi un increment al mărimii unghiului Δθ=0.005 rad. În suprafeței elicoidale măsurate

tabelul 9. 2, sunt prezentate coordonate ale punctelor aparținând suprafeței elicoidale efectiv măsurată.

NOTĂ

Parametrul elicoidal se obține în baza măsurării unghiului β_{ex} [vezi (9.15)].

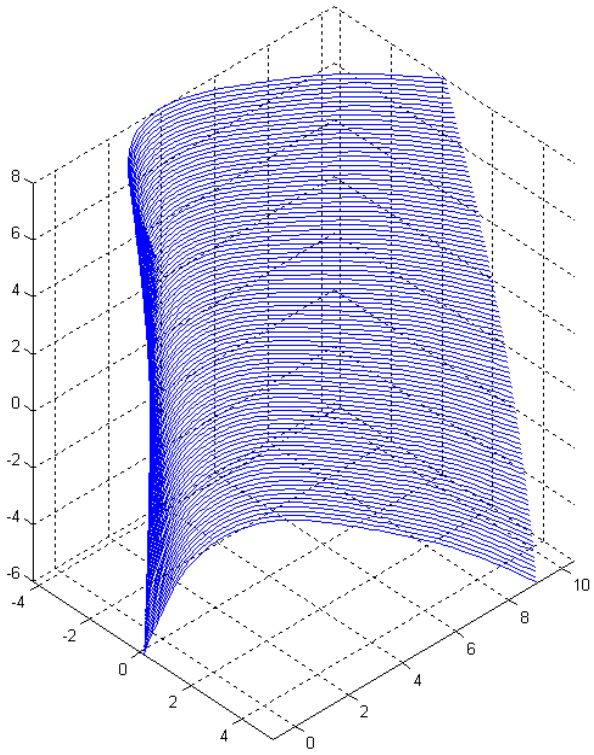


Figura 9. 10. Reprezentarea în formă discretă a suprafeței elicoidale măsurate

Tabelul 9. 2. Coordonate ale modelului discret al suprafeței măsurate.

X	Y	Z
-0.754	0.35493	1.000
-0.6456	0.2083	1.000
-0.5372	0.077372	1.000
-0.4288	-0.04263	1.000
-0.3204	-0.15541	1.000
-0.212	-0.26382	1.000
-0.1036	-0.36991	1.000
0.0048	-0.47517	1.000
0.1132	-0.58055	1.000
0.2216	-0.68659	1.000
0.33	-0.79354	1.000
0.4384	-0.90137	1.000
0.5468	-1.0099	1.000
0.6552	-1.1188	1.000
0.7636	-1.2276	1.000
0.872	-1.3359	1.000
0.9804	-1.4431	1.000
1.0888	-1.5489	1.000
1.1972	-1.6528	1.000
1.3056	-1.7543	1.000
1.414	-1.853	1.000
1.5224	-1.9488	1.000
1.6308	-2.0412	1.000
1.7392	-2.1302	1.000
1.8476	-2.2155	1.000
1.956	-2.297	1.000
2.0644	-2.3748	1.000
2.1728	-2.4487	1.000
2.2812	-2.5188	1.000
2.3896	-2.5851	1.000
2.498	-2.6478	1.000
2.6064	-2.7067	1.000
2.7148	-2.7622	1.000
2.8232	-2.8141	1.000
2.9316	-2.8625	1.000
3.04	-2.9076	1.000
3.1484	-2.9494	1.000
3.2568	-2.9879	1.000
3.3652	-3.023	1.000
3.4736	-3.0548	1.000
3.582	-3.0831	1.000
3.6904	-3.1081	1.000
3.7988	-3.1294	1.000
3.9072	-3.1471	1.000

4.0156	-3.161	1.000
4.124	-3.1709	1.000
4.2324	-3.1766	1.000
4.3408	-3.1781	1.000
4.4492	-3.1751	1.000
4.5576	-3.1675	1.000
4.666	-3.1551	1.000
4.7744	-3.1377	1.000
4.8828	-3.1153	1.000
4.9912	-3.0876	1.000
5.0996	-3.0547	1.000
5.208	-3.0164	1.000
5.3164	-2.9727	1.000
5.4248	-2.9237	1.000
5.5332	-2.8693	1.000
5.6416	-2.8098	1.000
5.75	-2.7451	1.000
5.8584	-2.6755	1.000
5.9668	-2.6011	1.000
6.0752	-2.5223	1.000
6.1836	-2.4392	1.000
6.292	-2.3522	1.000
6.4004	-2.2616	1.000
6.5088	-2.1677	1.000
6.6172	-2.071	1.000
6.7256	-1.9716	1.000
6.834	-1.87	1.000
6.9424	-1.7664	1.000
7.0508	-1.6612	1.000
7.1592	-1.5547	1.000
7.2676	-1.4468	1.000
7.376	-1.338	1.000
7.4844	-1.228	1.000
7.5928	-1.1171	1.000
7.7012	-1.005	1.000
7.8096	-0.89155	1.000
7.918	-0.77642	1.000
8.0264	-0.65923	1.000
8.1348	-0.53943	1.000
8.2432	-0.41643	1.000
8.3516	-0.28952	1.000
8.46	-0.15789	1.000
8.5684	-0.02069	1.000
8.6768	0.12296	1.000
8.7852	0.27398	1.000
8.8936	0.43324	1.000
9.002	0.60156	1.000
9.1104	0.77962	1.000

9.2188	0.9679	1.000
9.3272	1.1666	1.000
9.4356	1.3757	1.000
9.544	1.5944	1.000
9.6524	1.8217	1.000
9.7608	2.0556	1.000
9.8692	2.2934	1.000
9.9776	2.5313	1.000
10.086	2.7642	1.000

Această formă de exprimare a suprafeței permite profilarea sculelor mărginite de suprafețe primare de revoluție (pentru cazul concret, scula disc) reciproc înfăşurătoare suprafeței elicoidale.

9.1.2. Formă de reprezentare a flancului elicoidal al dintelui unei roți dințate evolventice

Pe maşina de măsurat 3D, vezi figura 9. 11, se măsoară generatoarea flancului evolventic, tabelul 9. 3.

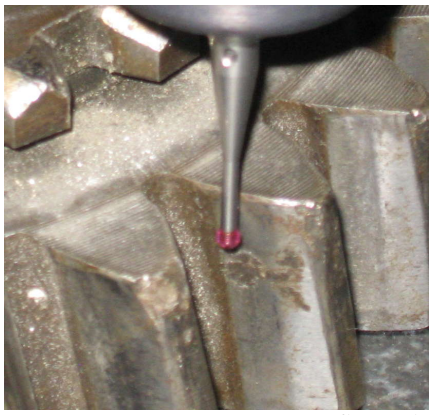


Figura 9. 11. Masurarea generatoarei flancului evolventic
(z= constant)

Tabelul 9. 3 Generatoarea flancului evolventic

Line ***j***	**Crt. no.**	**X_1 [mm]**	**Y_1 [mm]**	**Z_1 [mm]**
1	**1**	-8.407	-75.045	-2.000
	2	-6.501	-73.030	-2.000
	3	-5.082	-70.822	-2.000
	4	-3.963	-68.659	-2.000
	5	-2.955	-66.306	-2.000

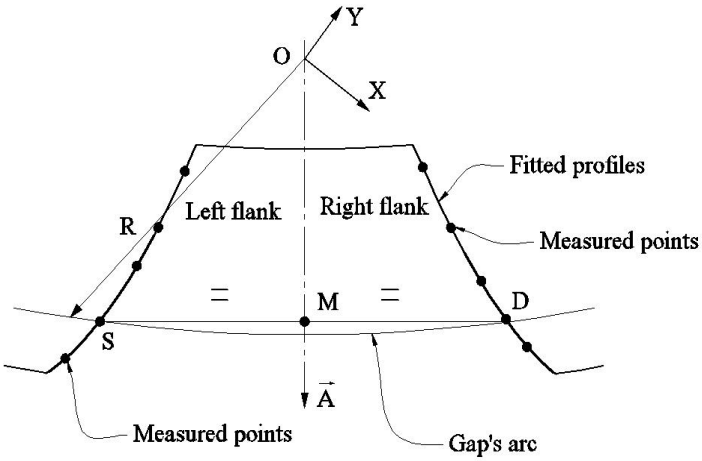


Figura 9. 12. Schema punctelor măsurate pe generatoarea flancului evolventic, Z_I=constant

De asemenea, se măsoară pe maşina de măsurat profiluri, figura 9. 12, unghiul de înclinare a elicei flancului evolventic pe diametrul exterior. Se calculează, vezi forma (9.13), mărimea parametrului elicoidal p= 322.8 mm.

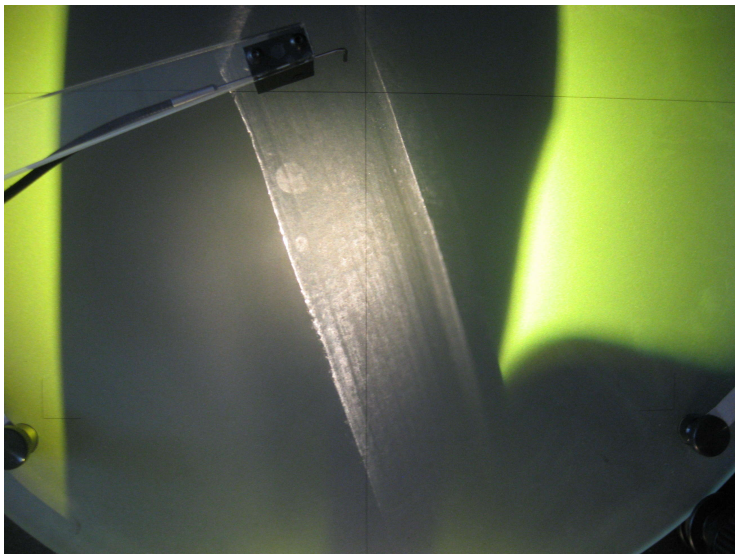


Figura 9. 13. Măsurarea unghiului β_{ex} al flancului evolventic

În tabelul 9. 4 şi figura 9. 14, se prezintă forma polinomului de substituire a generatoarei construită discret.

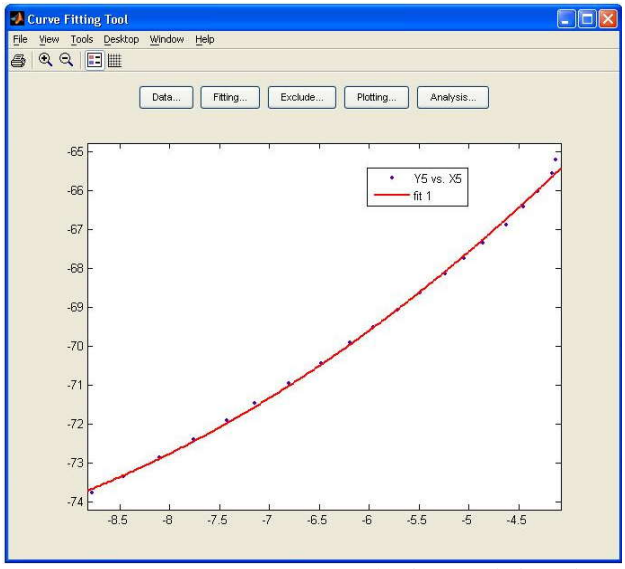


Figura 9. 14. Forma generatoarei (polinomul de substituire)

Tabelul 9. 4. Coordonate ale generatoarei substituite

Line *j*	**Crt. no.**	**X_1 [mm]**	**Y_1 [mm]**	**Z_1 [mm]**
1	**1**	-8.500	-76.179	-2.000
	2	-8.400	-76.165	-2.000
	⋮	⋮	⋮	⋮
	60	-2.600	-75.131	-2.000
	61	-2.500	-75.107	-2.000

În baza transformării (9.18),se reprezintă, în figura 9. 15 şi tabelul 9. 5, forma flancului evolventic, aproximat punct cu punct, şi coordonate ale flancului evolventic cunoscut în formă discretă.

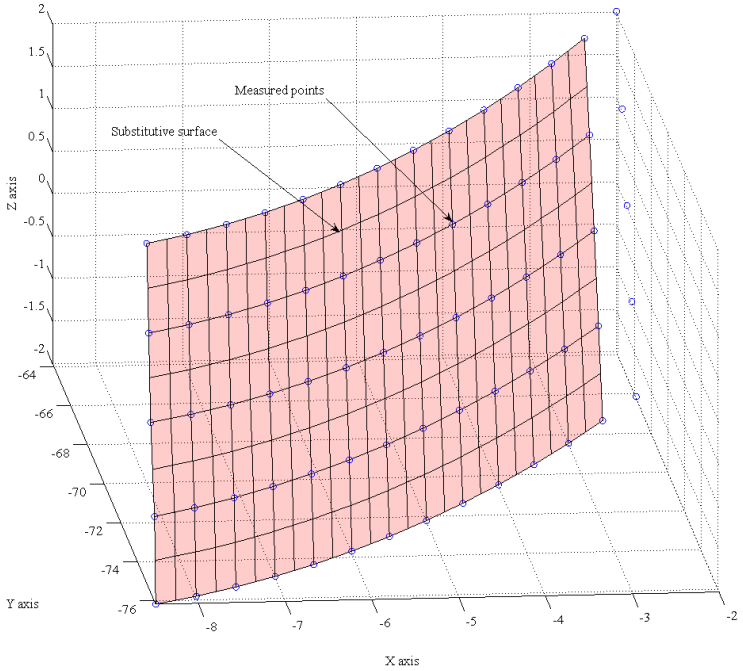


Figura 9. 15. Reprezentarea flancului elicoidal, punct cu punct

Tabelul 9. 5. Suprafața elicoidală punct cu punct

Line *j*	Crt. no.	X_1 [mm]	Y_1 [mm]	Z_1 [mm]
1	**1**	-8.500	-76.179	-2.000
	2	-8.400	-76.165	-2.000
	⋮	⋮	⋮	⋮
	60	-2.600	-75.131	-2.000
	61	-2.500	-75.107	-2.000
2	**1**	-8.500	-76.107	-1.900
	2	-8.400	-76.089	-1.900
		⋮	⋮	⋮
	60	-2.600	-75.051	-1.900
	61	-2.500	-75.027	-1.900
⋮	⋮	⋮	⋮	⋮
41	**1**	-8.500	-67.358	2.000
	2	-8.400	-67.308	2.000
	⋮	⋮	⋮	⋮
	60	-2.600	-65.236	2.000
	61	-2.500	-65.175	2.000

9.1.3. Formă de reprezentare a flancului elicoidal complex al dintelui unui rotor de compresor elicoidal

În practica curentă, pot apărea situații în care profilurile suprafețelor elicoidale sunt profiluri complexe formate din arce de curbe ce pot fi descrise prin ecuații analitice combinate cu profiluri descrise prin coordonate punct cu punct, care pot fi aproximate cu polinoame Bezier.

În cele ce urmează, se prezintă o aplicație în sensul celor prezentate anterior, pentru suprafețele active ale rotorului compresorului elicoidal.

Profilul cremalierei generatoare

Se consideră că profilul frontal al melcilor compresorului elicoidal rezultă ca înfăşurătoare a profilului cremalierei generatoare (cremaliera are o formă impusă, care satisface cerințele specifice unei construcții a melcilor compresorului elicoidal: lipsa punctelor singulare pe profil; asimetrie a profilului generator; linie de angrenare închisă şi de lungime minimă).

În figura 9. 16, este prezentată forma impusă a profilului transversal al cremalierei generatoare, a cărui înfăşurătoare este profilul transversal al rotorilor compresorului elicoidal.

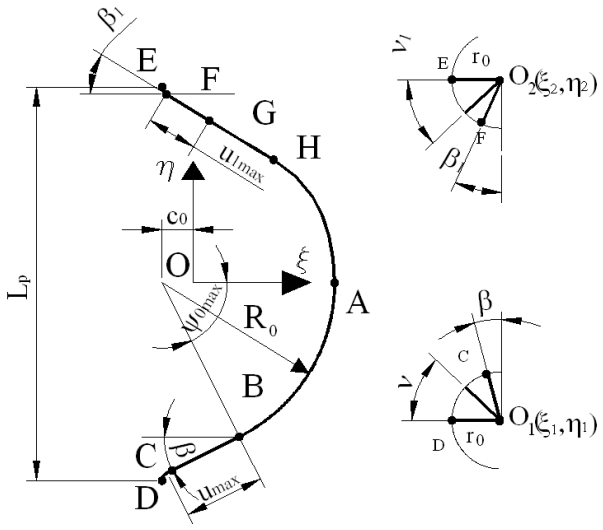


Figura 9. 16. Profilul transversal al cremalierei generatoare.
Sisteme de referință.

Alegerea formei cremalierei generatoare este necesar să conducă la forme ale profilurilor transversale ale rotoarelor de compresor care să asigure:

- o accentuată asimetrie a formei profilului, în vederea asigurării unui debit satisfăcător al compresorului;
- o linie de contact între rotoarele compresorului închisă, în vederea asigurării etanşeităţii camerelor de comprimare;
- un volum încastrat între lobii rotoarelor cât mai redus;
- să asigure inexistența punctelor singulare pe profilurile transversale;
- o tehnologicitate cât mai bună a rotoarelor compresorului, prin asigurarea unor profiluri ale sculelor generatoare fără discontinuități

Se acceptă un profil complex al cremalierei generatoare, care este format din ansamblul de profiluri elementare:

Flancul din stânga:

AB –arc de cerc de rază R_0;
BC –segment de dreaptă;
CD – arc de cerc de rază r_0.

Flancul din dreapta:

AH – arc de curbă (polinom Bezier de gradul II);
HG – arc de curbă (polinom Bezier de gradul II);
GF – segment de dreaptă;
FE – arc de cerc de rază r_0.

Ecuațiile parametrice ale profilurilor elementare ale cremalierei

•*Arcul de cerc AB, de raza R_0*

Ecuațiile parametrice, în sistemul de referință al cremalierei, în funcție de parametrul variabil unghiular ψ:

$$\begin{cases} \xi(\psi) = R_0 \cdot \cos\psi - c_0; \\ \eta(\psi) = -R_0 \cdot \sin\psi; \\ \zeta = 0. \end{cases} \tag{9.26}$$

Se impun mărimile $R_{0,}$ c_0, şi ψ_{max}, vezi figura 9. 16.

• *Segmentul de dreaptă BC*

Ecuaţiile parametrice în sistemul cremalierei, funcţie de parametrul variabil ,u, sunt:

$$\begin{cases} \xi(u) = \xi_B - u \cdot \cos\beta; \\ \eta(u) = \eta_B + u \cdot \sin\beta; \\ \zeta = 0, \end{cases} \tag{9.27}$$

şi

$$\beta = \frac{\pi}{2} - \psi_{max} \tag{9.28}$$

.

Parametrul u variază între u_0=0 şi o valoare maximă,

$$u_{max} = \sqrt{(\xi_C - \xi_B)^2 + (\eta_C - \eta_B)^2}\ . \tag{9.29}$$

Se cunosc $R_{0,}$ c_0, ψ_{max} şi coordonatele punctului B :

$$\xi_B = R_0 \cdot \cos\psi_{max};\ \eta_B = -R_0 \cdot \sin\psi_{max}\ . \tag{9.30}$$

Coordonatele punctului C se determină din considerentul ca segmentul de dreaptă BC să fie tangent cercului cu centrul în O_1 şi de rază r_0.

• *Arcul de racordare $\widehat{CD}$, de rază r_0*

Ecuaţiile parametrice în sistemul cremalierei, în funcţie de parametrul *v,* sunt:

$$\begin{cases} \xi(\nu) = r_0 \cdot \cos\nu_1 + \xi_{0_1}; \\ \eta(\nu) = -r_0 \cdot \sin\nu_1 + \eta_{0_1}. \end{cases} \tag{9.31}$$

Se presupun cunoscute coordonatele centrului cercului, $O_1(\xi_{O_1};\eta_{O_1})$.

• *Arcul de racordare EF, de rază r_0*

În mod similar, ecuaţiile parametrice ale arcului de cerc EF, în funcţie de parametrul variabil ν_1, sunt:

$$\begin{cases} \xi(\nu_1) = r_0 \cdot \cos\nu_1 + \xi_{0_2}; \\ \eta(\nu_1) = -r_0 \cdot \sin\nu_1 + \eta_{0_2}. \end{cases} \tag{9.32}$$

Lungimea pasului cremalierei se calculează cu relaţia

$$L_p = \frac{2\pi \cdot Rr_1}{6}. \tag{9.33}$$

Se impun

$$\xi_{0_2} = \xi_{0_1}\ ,\ |\eta_{0_2}| = |\eta_{0_1}|. \tag{9.34}$$

• *Segmentul de dreaptă FG*

Ecuaţiile parametrice ale dreptei FG, în sistemul cremalierei, în funcţie de parametrul variabil u_1, sunt :

$$\begin{cases} \xi(u_1) = +u_1 \cdot \cos\beta_1 + \xi_F; \\ \eta(u_1) = -u_1 \cdot \sin\beta_1 + \eta_F. \end{cases} \tag{9.35}$$

Cum

$$\beta_1 = \frac{\pi}{2} - \nu_{1max}, \tag{9.36}$$

se pot exprima coordonatele punctului $F(\xi_F,\eta_F)$ în forma:

$$\begin{cases}\xi_F = \xi_{0_2} - r_0 \cdot \sin\nu_{1max};\\ \eta_F = \eta_{0_2} - r_0 \cdot \cos\nu_{1max};\end{cases} \tag{9.37}$$

$$tg\beta_1 = \frac{|\eta_G - \eta_F|}{|\xi_G - \xi_F|}. \tag{9.38}$$

Parametrul u_1 variază între $u_{1min}=0$ şi o valoare maximă,

$$u_{1max} = \sqrt{(\xi_G - \xi_F)^2 + (\eta_G - \eta_F)^2}\ . \tag{9.39}$$

Se cunosc $R_{0,}\ c_0,\ \psi_0$ şi coordonatele punctului $F(\xi_F,\eta_F)$.

• *Curbele AH şi HG*

Se propune o formă a profilului pentru arcul $\widehat{AHG}$ un ansamblu de curbe Bézier, - polinoame de gradul II – care trebuie să îndeplinească condiţii geometrice de continuitate cu celelalte curbe elementare constituente ale profilului cremalierei.

Ecuaţii ale polinomului de substituire Bézier ale arcului $\widehat{AH}$:

$$\begin{cases}P_{\xi AH} = \lambda_1^2 A_\xi + 2(1-\lambda_1)\lambda_1 B_\xi + (1-\lambda_1)^2 C_\xi;\\ P_{\eta AG} = \lambda_1^2 A_\eta + 2(1-\lambda_1)\lambda_1 B_\eta + (1-\lambda_1)^2 C_\eta,\end{cases}. \tag{9.40}$$

cu $0 \le \lambda_1 \le 1$.

Polinomul Bézier substitutiv al arcului $\widehat{HG}$:

$$\begin{cases}P_{\xi HG} = \lambda_2^2 D_\xi + 2(1-\lambda_2)\lambda_2 E_\xi + (1-\lambda_2)^2 F_\xi;\\ P_{\eta HG} = \lambda_2^2 D_\eta + 2(1-\lambda_2)\lambda_2 E_\eta + (1-\lambda_2)^2 F_\eta,\end{cases} \tag{9.41}$$

cu $0 \le \lambda_2 \le 1$.

Se definesc derivatele de ordinul întâi ale polinoamelor:

$$\begin{cases}P'_{\xi AH} = 2\lambda_1 A_\xi + 2(1-2\lambda_1)B_\xi - 2(1-\lambda_1)C_\xi;\\ P'_{\eta AH} = 2\lambda_1 A_\eta + 2(1-2\lambda_1)B_\eta - 2(1-\lambda_1)C_\eta;\end{cases} \tag{9.42}$$

$$\begin{cases}P'_{\xi AG} = 2\lambda_2 D_\xi + 2(1-2\lambda_2)E_\xi - 2(1-\lambda_2)F_\xi;\\ P'_{\eta AG} = 2\lambda_2 D_\eta + 2(1-2\lambda_2)E_\eta - 2(1-\lambda_2)F_\eta.\end{cases} \tag{9.43}$$

Condiţiile de continuitate a profilurilor elementare constituente ale profilului cremalierei generatoare impun determinarea coeficienţilor polinoamelor Bézier astfel încât, în punctele de contact, să se definească o normală unică la cele două profiluri succesive.

Condiţii impuse celor două curbe:

● Condiţii de coincidenţă, în punctul A, a arcului AB cu AH ($\lambda_1=1$):

$$\begin{cases}\xi_A = P_{\xi AH};\\ \eta_A = P_{\eta AH}.\end{cases} \tag{9.44}$$

●Normala comună la cele două profiluri, în punctul A de contact ($\lambda_1=1$), impune îndeplinirea condiţiilor:

$$\begin{cases}\xi'_A = P'_{\xi AH};\\ \eta'_A = P'_{\eta AH}.\end{cases} \tag{9.45}$$

● În mod similar, coincidența, în punctul G, a segmentelor FG si HG (λ_2=0) conduce la:

$$\begin{cases}\xi_A = P_{\xi HG};\\ \eta_A = P_{\eta HG}.\end{cases} \tag{9.46}$$

● Normala comună la cele două profiluri în punctul G conduce la (λ_2=0):

$$\begin{cases}\xi'_G = P'_{\xi HG};\\ \eta'_G = P'_{\eta HG}.\end{cases} \tag{9.47}$$

De asemenea, în poziția λ_1=0 şi λ_2=1 (punctul H), cele două polinoame trebuie să fie identice, adică:

- condiția de punct comun ,

$$\begin{cases}P_{\xi AH(\lambda_1=0)} = C_\xi = P_{\xi HG(\lambda_2=1)} = D_\xi;\\ P_{\eta AH(\lambda_1=0)} = C_{\eta(\lambda_2=1)} = P_{\eta HG(\lambda_2=1)} = D_\eta;\end{cases} \tag{9.48}$$

- condiția de normală comună,

$$\begin{cases}P'_{\xi AH(\lambda_1=0)} = 2(B_\xi - C_\xi) = P'_{\xi HG(\lambda_2=1)} = 2(D_\xi - E_\xi);\\ P_{\eta AH(\lambda_1=0)} = 2(B_\eta - C_\eta) = P'_{\eta HG(\lambda_2=1)} = 2(D_\eta - E_\eta).\end{cases} \tag{9.49}$$

Ținând seama de definiția coordonatelor punctului A,

$$\begin{cases}\xi_A = R_0 - c_0;\\ \eta_A = 0,\end{cases} \tag{9.50}$$

rezultă coeficienții:

$\begin{cases}A_\xi = R_0 - c_0;\\ A_\eta = 0,\end{cases}$ în funcție de mărimile constructive ale profilului cremalierei.

Pe de altă parte, contactul polinomului substitutiv al arcului $\widehat{AH}$, cu arcul $\widehat{AB}$, în punctul A, impune condițiile:

$$\begin{cases}\xi(\psi) = R_0\cos\psi - c_0;\\ \eta(\psi) = -R_0\sin\psi;\end{cases} \tag{9.51}$$

$$\begin{cases}\xi'_{(\psi)} = -R_0\sin\psi;\\ \eta'_{(\psi)} = -R_0\cos\psi.\end{cases} \tag{9.52}$$

În punctul A,

$$\psi=0, \Rightarrow \begin{cases}\xi'_A = 0;\\ \eta'_A = -R_0,\end{cases} \tag{9.53}$$

reprezentând parametrii directori ai tangentei la profilul circular şi, din egalitățile:

$$\begin{cases}2\lambda_1 A_\xi + 2(1-2\lambda_1)B_\xi - 2(1-\lambda_1)C_\xi = 0;\\ 2\lambda_1 A_\eta + 2(1-2\lambda_1)B_\eta - 2(1-\lambda_1)C_\eta = -R_0,\end{cases} \tag{9.54}$$

pentru λ_1=1, în punctul A, se obțin relațiile între coeficienți:

$$\begin{cases}A_\xi = B_\xi;\\ A_\eta - B_\eta = -R_0.\end{cases} \tag{9.55}$$

Similar, pentru punctul G, se definesc condițiile:

$$\begin{cases}\xi_G = \xi_F + u_{1max} \cdot \cos\nu_{1max}; \\ \eta_G = \eta_F - u_{1max} \cdot \sin\nu_{1max}, \end{cases} \tag{9.56}$$

rezultând

$$\begin{cases}\xi_G = F_\xi, \\ \eta_G = F_\eta. \end{cases} \tag{9.57}$$

De asemenea, din aceleaşi considerente, pentru punctul G, rezultă:

$$\begin{cases}\xi(u_1) = \xi_F + u_1\cos\beta_1; \\ \eta(u_1) = \eta_F - u_1\sin\beta_1; \end{cases} \tag{9.58}$$

$$\begin{cases}\xi'_{(u_1)} = \cos\beta_1; \\ \eta'_{(u_1)} = -\sin\beta_1; \end{cases} \tag{9.59}$$

$$\begin{cases}2(D_\xi - E_\xi) = \cos\beta_1; \\ 2(D_\eta - E_\eta) = -\sin\beta_1. \end{cases} \tag{9.60}$$

Ansamblul de ecuaţii format din (9.48), (9.49), (9.51), (9.55), (9.57), (9.60), permite exprimarea celor 12 necunoscute, mărimile coeficienţilor polinoamelor substitutive arcelor $\widehat{AH}$ şi $\widehat{HG}$, în funcţie de elementele anterior definite:

$$\begin{cases} A_\xi = R_0 - c_0; \\ A_\eta = 0; \\ B_\xi = A_\xi; \\ B_\eta = R_0/2; \\ C_\xi = D_\xi; \\ C_\eta = D_\eta; \\ E_\xi = F_\xi + 0.5\cos\beta_1; \\ E_\eta = F_\xi - 0.5\sin\beta_1; \\ F_\xi = \xi_G; \\ F_\eta = \eta_G; \\ D_\xi = 0.5 \cdot (B_\xi + E_\xi); \\ D_h = 0.5 \cdot (B_\eta + E_\eta). \end{cases} \tag{9.61}$$

Determinarea profilurilor transversale ale rotoarelor

Odată definită forma analitică a cremalierei generatoare a profilurilor transversale a rotoarelor, condus şi conducător, se poate determina, în baza legilor fundamentale ale înfăşurării profilurilor, forma profilurilor transversale ale rotoarelor. Se propune o rezolvare a problemei utilizând „metoda normalelor” (Willis).

Evident, se pot utiliza şi teoremele fundamentale ale înfăşurării suprafeţelor (teorema I Olivier, teorema Gohman) sau teoreme complementare precum „metoda distanţei minime” sau „metoda traiectoriilor plane de generare”.

Determinarea profilului transversal
al melcului conducător

În figura 9. 17, sunt prezentate sistemele de referință şi mişcările de generare; vezi şi figura 9. 16.

Condiția ca profilurile elementare ale cremalierei să admită o înfăşurătoare, conform metodei normalelor, este ca normala la profil să intersecteze centroida asociată profilului, aici centroida C.

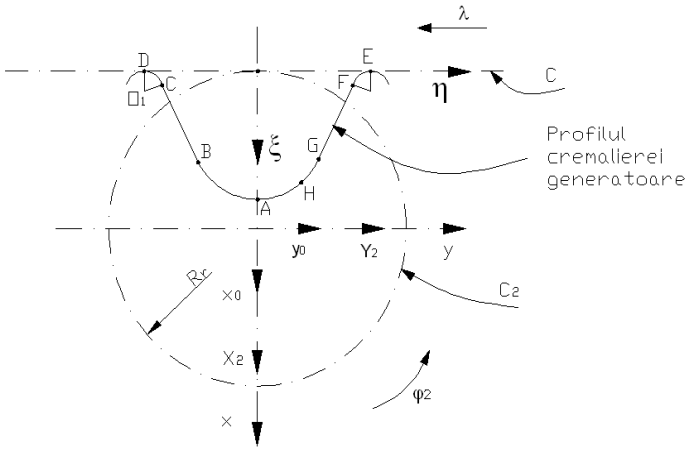


Figura 9. 17. Generarea profilului rotorului conducător

•*Arcul de cerc* $\overset{\frown}{AB}$

Normala la profilul circular $\overset{\frown}{AB}$, principial, are forma

$$N_{AB}:\left[\xi-\xi(\psi)\right]\xi'_{(\psi)}+\left[\eta-\eta(\psi)\right]\eta'_{(\psi)}=0. \tag{9.62}$$

Ecuația centroidei asociată cremalierei, dreapta C, vezi figura 9. 3, are ecuațiile:

$$(C)\begin{cases}\xi=0;\\ \eta=\lambda,\end{cases} \tag{9.63}$$

λ parametru variabil.

Condiția de intersecție a normalei (9.62) cu centroida (9.63) reprezintă condiția de înfăşurare,

$$\lambda=\frac{\eta(\psi)\cdot\eta'(\psi)+\xi(\psi)\cdot\xi'(\psi)}{\eta'(\psi)}. \tag{9.64}$$

În condiția (9.64), cu $\eta'(\psi)$, $\xi'(\psi)$ s-au notat derivatele parțiale în raport cu parametrul variabil, generic, ψ.

Din ecuația ce descrie mişcarea relativă a sistemelor de referință $X_1Y_1Z_1$ si $\xi\eta\zeta$ (9.65)

$$X_1=\omega_3(-\varphi_1)\cdot\left[\xi+\begin{Vmatrix}Rr_1\\ -Rr_1\cdot\varphi_1\\ 0\end{Vmatrix}\right] \tag{9.66}$$

rezultă:

$$\begin{cases}X_1=R_0\cdot\cos(\psi+\varphi_1)+(Rr_1-c_0)\cdot\cos\varphi_1+Rr_1\cdot\varphi_1\cdot\sin\varphi_1;\\ Y_1=-R_0\cdot\sin(\psi+\varphi_1)-(Rr_1-c_0)\cdot\sin\varphi_1+Rr_1\cdot\varphi_1\cdot\cos\varphi_1;\\ Z_1=0.\end{cases} \tag{9.67}$$

Ecuațiile (9.67) reprezintă familia de curbe $\left(\Sigma_{AB}\right)_{\varphi_1}$, generată de arcul de cerc $\overset{\frown}{AB}$, în sistemul de coordonate al melcului conducător, $X_1Y_1Z_1$. Profilul melcului conducător se obține din condiția ca acesta să fie în înfășurare cu profilul cremalierei de referință.

Substituind în (9.66), se obține condiția de înfășurare specifică,

$$\varphi_1 = -\frac{c_0}{Rr_1}\operatorname{tg}\psi\,. \tag{9.68}$$

Ansamblul de ecuații (9.67) şi (9.68) reprezintă profilul melcului conducător în secțiune transversală, corespunzător arcului de cerc $\overset{\frown}{AB}$ al cremalierei.

• *Segmentul de dreaptă* $\overline{BC}$

Similar, cu cele de mai sus pentru segmentul de dreaptă BC, vezi (9.27), rezultă familia de profiluri:

$$\begin{cases} X_1 = -u\cdot\cos\left(\beta-\varphi_1\right)+\left(Rr_1+\xi_B\right)\cdot\cos\varphi_1-\left(-Rr_1\cdot\varphi_1+\eta_B\right)\cdot\sin\varphi_1; \\ Y_1 = u\cdot\sin\left(\beta-\varphi_1\right)+\left(Rr_1+\xi_B\right)\cdot\sin\varphi_1+\left(-Rr_1\cdot\varphi_1+\eta_B\right)\cdot\cos\varphi_1; \\ Z_1 = 0, \end{cases} \tag{9.69}$$

căreia i se asociază condiția de înfășurare

$$\lambda = -\frac{u}{\cos\psi_{max}}+\xi_B\cdot\operatorname{tg}\psi_{max}-\eta_B \tag{9.70}$$

sau, sub forma

$$\varphi_1 = \frac{-\dfrac{u}{\cos\psi_{max}}+\xi_B\cdot\operatorname{tg}\psi_{max}-\eta_B}{Rr_1}. \tag{9.71}$$

Profilul melcului conducător în secțiunea transversală este dat de ansamblul ecuațiilor (9.69) şi (9.71), corespunzător segmentului $\overline{BC}$ al cremalierei.

•*Arcul de racordare* $\overset{\frown}{CD}$*, de rază* r_0

Familia de profiluri:

$$\begin{cases} X_1 = -r_0\cdot\cos\left(\nu+\varphi_1\right)+\left(Rr_1+\xi_{0_1}\right)\cdot\cos\varphi_1+\left(Rr_1\cdot\varphi_1+\eta_{0_1}\right)\cdot\sin\varphi_1; \\ Y_1 = r_0\cdot\sin\left(\nu+\varphi_1\right)-\left(Rr_1+\xi_{0_1}\right)\cdot\sin\varphi_1+\left(Rr_1\cdot\varphi_1+\eta_{0_1}\right)\cdot\cos\varphi_1; \\ Z_1 = 0; \end{cases} \tag{9.72}$$

împreună cu condiția de înfășurare

$$\varphi_1 = \frac{\xi_{0_1}\cdot\operatorname{tg}\nu+\eta_{0_1}}{Rr_1}, \tag{9.73}$$

(cu v şi φ_1 - variabile); $\nu_{1max} = \dfrac{\pi}{2}-\beta_1$, vezi şi figura 9. 16.

Profilul melcului conducător, în secțiunea transversală, este dat de ansamblul ecuațiilor (9.71) şi (9.72).

•*Arc de cerc,* $\widehat{EF}$.

Familia de profiluri:

$$\begin{cases} X_1 = -r_0 \cdot \cos\left(-\nu_1 + \varphi_1\right) + \left(Rr_1 + \xi_{0_2}\right) \cdot \cos\varphi_1 + \left(Rr_1 \cdot \varphi_1 - \eta_{0_2} + L_p\right) \cdot \sin\varphi_1; \\ Y_1 = r_0 \cdot \sin\left(-\nu_1 + \varphi_1\right) - \left(Rr_1 + \xi_{0_2}\right) \cdot \sin\varphi_1 + \left(Rr_1 \cdot \varphi_1 - \eta_{0_2} + L_p\right) \cdot \cos\varphi_1; \\ Z_1 = 0, \end{cases} \tag{9.74}$$

împreună cu condiția specifică de înfășurare

$$\varphi_1 = \frac{\xi_{0_2} \cdot \mathrm{tg}\nu_1 + \eta_{0_2}}{Rr_1}, \text{ cu } \nu_1 \text{ şi } \varphi_1 - \text{variabile.} \tag{9.75}$$

Profilul melcului conducător din secțiunea transversală este dat de ansamblul ecuațiilor (9.74) şi (9.75), pentru limitele de variație ale parametrului ν_1, $0 \le \nu_1 \le (\pi/2 - \beta)$.

• *Segmentul de dreaptă* $\overline{FG}$.

Familia de profiluri generată de segmentul de dreaptă $\overline{FG}$:

$$\begin{cases} X_1 = u_1 \cdot \cos\left(\varphi_1 + \beta_1\right) + \left(Rr_1 + \xi_F\right) \cdot \cos\varphi_1 + \left(Rr_1 \cdot \varphi_1 + \eta_F\right) \cdot \sin\varphi_1; \\ Y_1 = -u_1 \cdot \sin\left(\varphi_1 + \beta_1\right) - \left(Rr_1 + \xi_F\right) \cdot \sin\varphi_1 + \left(Rr_1 \cdot \varphi_1 + \eta_F\right) \cdot \cos\varphi_1; \\ Z_1 = 0, \end{cases} \tag{9.76}$$

împreună cu (9.77)- condiția specifică de înfăşurare,

$$\varphi_1 = \frac{-\dfrac{u_1}{\sin\beta_1} + \xi_F \cdot \mathrm{ctg}\beta_1 + \eta_F}{Rr_1}, \tag{9.77}$$

determină profilul melcului conducător, în secțiunea transversală.

• *Curbele* $\widehat{AH}$ *şi* $\widehat{HG}$

Pentru forma polinoamelor Bézier:

$$\widehat{AH} \begin{cases} \xi\left(\lambda_1\right) = \lambda_1^2 A_1 + 2\left(1-\lambda_1\right)\lambda_1 B_1 + \left(1-\lambda_1\right)^2 C_1; \\ \eta\left(\lambda_1\right) = \lambda_1^2 A_2 + 2\left(1-\lambda_1\right)\lambda_1 B_2 + \left(1-\lambda_1\right)^2 C_2; \end{cases} \tag{9.78}$$

$$\widehat{GH} \begin{cases} \xi\left(\lambda_2\right) = \lambda_2^2 D_1 + 2\left(1-\lambda_2\right)\lambda_2 E_1 + \left(1-\lambda_2\right)^2 F_1; \\ \eta\left(\lambda_1\right) = \lambda_2^2 D_2 + 2\left(1-\lambda_2\right)\lambda_2 E_2 + \left(1-\lambda_2\right)^2 F_2, \end{cases}$$

se determină forma principială a familiei de profiluri descrise în sistemul de referință al rotorului conducător:

$$\begin{cases} X_1 = \left(\xi\left(\lambda_{1,2}\right) - Rr_1\right) \cdot \cos\varphi_1 + \left(\eta\left(\lambda_{1,2}\right) + Rr \cdot \varphi_1\right) \cdot \sin\varphi_1; \\ Y_1 = -\left(\xi\left(\lambda_{1,2}\right) - Rr_1\right) \cdot \sin\varphi_1 + \left(\eta\left(\lambda_{1,2}\right) + Rr_1 \cdot \varphi_1\right) \cdot \cos\varphi_1; \\ Z_2 = 0, \end{cases} \tag{9.79}$$

cu φ_1 şi λ_1, λ_2 - variabile independente.

Principial, profilul melcului conducător în secţiune transversală este dat de ansamblul ecuaţiilor (9.67), (55) şi condiţia de înfăşurare specifică scrisă în baza teoremei traiectoriilor plane de înfăşurare,

$$\frac{X'_{1_\lambda}}{X'_{1_{\varphi 2}}} = \frac{Y'_{1_\lambda}}{Y'_{1_{\varphi 2}}}, \quad (\lambda = \lambda_{1,2}). \tag{9.80}$$

Termenii din (9.80) sunt derivatele parţiale ale familiei (9.79).

SUPRAFEŢELE ELICOIDALE PERIFERICE ALE ROTORILOR

Suprafeţele elicoidale ale rotoarelor melcilor de compresor, condus şi conducător, sunt suprafeţe elicoidale cilindrice de pas constant, de sensuri diferite.

Ca urmare, determinarea formelor analitice ale flancurilor melcilor se realizează prin imprimarea unei mişcări elicoidale a secţiunii transversale a melcului în jurul axei de rotaţie a acestuia – axa axoidei asociată suprafeţei elicoidale,

$$\begin{Vmatrix} X_1 \\ Y_1 \\ Z_1 \end{Vmatrix} = \omega_3^T(\theta_1) \cdot \begin{Vmatrix} X_1(u) \\ Y_1(u) \\ 0 \end{Vmatrix} + \begin{Vmatrix} 0 \\ 0 \\ p \cdot \theta_1 \end{Vmatrix}, \tag{9.81}$$

melc dreapta, cu $X_1(u)$, $Y_1(u)$, generic, forma ecuaţiilor secţiunii transversale a melcului (vezi (9.67) ...(9.79)).

După dezvoltări, rezultă forma :

$$\begin{cases} X_1 = X_1(u)\cdot\cos\theta_1 - Y_1(u)\cdot\sin\theta_1; \\ Y_1 = X_1(u)\cdot\sin\theta_1 - Y_1(u)\cdot\cos\theta_1; \\ Z_1 = p_1\cdot\theta_1, \end{cases} \tag{9.82}$$

cu p_1 – parametrul elicoidal (melc dreapta).

• Suprafeţele elicoidale ale lobilor melcului conducător

Din (9.57), pentru profilurile secţiunii frontale ale melcului conducător (9.67)...(9.79) rezultă:

• Pentru Arcul $\widehat{AB}$:

$$\begin{cases} X_1 = R_0\cdot\cos(\psi+\varphi_1-\theta_1) + (Rr_1 - c_0)\cdot\cos(\varphi_1-\theta_1) + Rr_1\cdot\varphi_1\cdot\sin(\varphi_1-\theta_1); \\ Y_1 = R_0\cdot\sin(\psi+\varphi_1-\theta_1) + (Rr_1 - c_0)\cdot\sin(\varphi_1-\theta_1) + Rr_1\cdot\varphi_1\cdot\cos(\varphi_1-\theta_1); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = -\dfrac{c_0\cdot tg\psi}{Rr_1}. \end{cases} \tag{9.83}$$

Sistemul de ecuaţii (9.83) reprezintă suprafaţa elicoidală periferică generată de arcul $\widehat{AB}$, pentru lobul conducător, cu φ_1 şi θ_1 parametrii unghiulari variabili.

• Suprafaţa elicoidală corespunzătoare segmentului de dreaptă $\overline{BC}$

Profilul melcului conducător, corespunzător segmentului de dreaptă $\overline{BC}$, conduce la suprafaţa elicoidală de ecuaţii:

$$\begin{cases} X_1 = u\cdot\sin\left(\varphi_1 - \psi_{max} - \theta_1\right) + \left(Rr_1 + \xi_B\right)\cdot\cos\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi_1 + \eta_B\right)\cdot\sin\left(\varphi_1 - \theta_1\right); \\ Y_1 = u\cdot\cos\left(\varphi_1 - \psi_{max} - \theta_1\right) - \left(Rr_1 + \xi_B\right)\cdot\sin\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi_1 + \eta_B\right)\cdot\cos\left(\varphi_1 - \theta_1\right); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = \dfrac{-\dfrac{u}{\cos\psi} + \xi_B\cdot tg\psi_{max} - \eta_B}{Rr_1}. \end{cases} \tag{9.84}$$

(φ_1 - condiția de înfășurare, parametrii variabili - u, φ_1, θ_1)

• ***Suprafața elicoidală corespunzătoare arcului*** $\overset{\frown}{CD}$ ***:***

$$\begin{cases} X_1 = -r_0\cdot\cos\left(\nu + \varphi_1 - \theta_1\right) + \left(Rr_1 + \xi_0\right)\cdot\cos\left(\varphi_1 - \theta_1\right) - \left(Rr_1\cdot\varphi_1 + \eta_{0_1}\right)\cdot\sin\left(\varphi_1 - \theta_1\right); \\ Y_1 = r_0\cdot\sin\left(\nu + \varphi_1 - \theta_1\right) - \left(Rr_1 + \xi_{r0}\right)\cdot\sin\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi_1 + \eta_{0_1}\right)\cdot\cos\left(\varphi_1 - \theta_1\right); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = \dfrac{-\xi_{0_1}\cdot tg\nu + \eta_{0_1}}{Rr_1}, \end{cases} \tag{9.85}$$

(condiția de înfășurare, cu parametrii variabili – v, φ_1, θ_1).

•***Suprafața elicoidală generată de arcul*** $\overset{\frown}{EF}$ ***:***

$$\begin{cases} X_1 = -r_0\cdot\cos\left(-\nu_1 + \varphi_1 - \theta_1\right) + \left(Rr_1 + \xi_{0_2}\right)\cdot\cos\left(\varphi_1 - \theta_1\right) - \left(Rr_1\cdot\varphi_1 - \eta_{0_2}\right)\cdot\sin\left(\varphi_1 - \theta_1\right); \\ Y_1 = r_0\cdot\sin\left(-\nu_1 + \varphi_1 - \theta_1\right) - \left(Rr_1 + \xi_{0_2}\right)\cdot\sin\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi - \eta_{0_2}\right)\cdot\cos\left(\varphi_1 - \theta_1\right); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = \dfrac{\xi_{0_2}\cdot tg\nu_1 + \eta_{0_2}}{Rr_1}. \end{cases} \tag{9.86}$$

(variabile –ν_1, θ_1, φ_1).

• ***Suprafața elicoidală generată de segmentuluil de dreaptă*** $\overline{FG}$ ***:***

$$\begin{cases} X_1 = u_1\cdot\cos\left(\varphi_1 + \beta_1 - \theta_1\right) + \left(Rr_1 + \xi_F\right)\cdot\cos\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi_1 + \eta_F\right)\cdot\sin\left(\varphi_1 - \theta_1\right); \\ Y_1 = -u_1\cdot\sin\left(\varphi_1 + \beta_1 - \theta_1\right) - \left(Rr_1 + \xi_F\right)\cdot\sin\left(\varphi_1 - \theta_1\right) + \left(Rr_1\cdot\varphi_1 + \eta_F\right)\cdot\cos\left(\varphi_1 - \theta_1\right); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = \dfrac{-\dfrac{u_1}{\sin\beta_1} + \xi_F\cdot ctg\beta_1 + \eta_F}{Rr_1}. \end{cases} \tag{9.87}$$

(variabile – u_1, φ_1, θ_1)

• ***Suprafața elicoidală generată de curbele Bezier,*** $\overset{\frown}{AH}$ ***şi*** $\overset{\frown}{HG}$ ***:***

Principial, suprafețele elicoidale corespunzătoare, au ecuații de forma:

$$\begin{cases} X_1 = \left(\xi(\lambda) + Rr_1\right)\cdot\cos\left(\varphi_1 - \theta_1\right) + \left(\eta(\lambda) + Rr_1\cdot\varphi_1\right)\cdot\sin\left(\varphi_1 - \theta_1\right); \\ Y_1 = -\left(\xi(\lambda) + Rr_1\right)\cdot\sin\left(\varphi_1 - \theta_1\right) + \left(\eta(\lambda) + Rr_1\cdot\varphi_1\right)\cdot\cos\left(\varphi_1 - \theta_1\right); \\ Z_1 = p_1\cdot\theta_1; \\ \varphi_1 = \varphi_1(\lambda). \end{cases} \tag{9.88}$$

φ_1 – condiția de înfășurare, vezi (9.65), cu λ generic şi variabilele (λ_1, λ_2, φ_1,θ_1).

9.2. Elaborarea de produse soft specifice

9.2.1. Produse soft pentru profilarea sculei cremalieră

Profilarea sculelor care generează prin înfăşurare prin metoda rulării – scula cremalieră şi scula roată – poate fi realizată, aşa cum este cunoscut, prin mai multe metode:

- metode analitice, în baza teoremelor fundamentale ale înfăşurării suprafeţelor, *teorema I Olivier*, *teorema Gohman*, *metoda normalelor*, *Willis*;
- metode analitice complementare, – *metoda „distanţei minime", metoda „familiei de cercuri substitutive", metoda „traiectoriilor plane de generare"*;
- metode grafo-analitice;
- metode grafice, utilizând facilităţile produselor soft de tip CAD,

Facem cuvenita menţiune că multitudinea de metode, propuse şi utilizate pentru studiul suprafeţelor (profilurilor) reciproc înfăşurătoare, respectă, evident, teorema fundamentală a înfăşurării. Soluţiile propuse, prin utilizarea acestor metode, conduc la rezultate foarte apropiate, în cele mai multe cazuri identice, ale formei profilurilor transversale ale sculelor, care generează prin înfăşurare vârtejuri ordonate de profiluri asociate unui cuplu de centroide în rulare.

Metoda cinematică în mediul grafic de proiectare CATIA

Se propune o nouă soluţie a problemei profilării sculei cremalieră reciproc înfăşurătoare a unui vârtej ordonat de profiluri (suprafeţe) asociat unui cuplu de centroide în rulare, făcând apel la facilităţile oferite de produsul soft CATIA, prin realizarea unei entităţi cinematice care să reproducă mişcarea de rulare a centroidelor: C_1 – cerc de rază R_{rp} , asociat vârtejului de profiluri de generat; C_2 – dreaptă, asociată spaţiului viitoarei scule cremalieră.

Soluţia cinematică grafică în mediul de proiectare CATIA, a fost imaginată ca un mecanism virtual de generare a traiectoriilor unor puncte în raport cu diferite sisteme de referinţă ale elementelor componente.

Rezolvarea propusă se bazează pe facilităţile mediului *Part* (*Part Environment*), în care se sintetizează elementele unui mecanism virtual capabil a simula condiţia de înfăşurare, în acest caz, condiţia normalelor. Aceste elemente, create în mediul *Part*, sunt introduse într-un fişier al mediului *Asembly*, asigurându-se poziţionarea elementelor mecanismului în poziţia de start, urmând ca, în mediul *DMU Kinematics* (*Digital Mock Up*), să se definească cuplele cinematice predefinite.

Rularea mecanismului se realizează prin comanda de simulare *Simulation*, stabilindu-se numărul de poziţii intermediare *Shots*, creându-se cu comanda *Replay* un film al poziţiilor succesive ale mecanismului.

Prin comanda *Trace*, se trasează traiectoria oricărui punct de pe un element al mecanismului, în raport cu oricare alt element al acestuia, inclusiv faţă de sistemul de referinţă fix, determinându-se astfel linia de angrenare între profilul de generat şi profilul sculei cremalieră.

Aceste traiectorii reprezintă curbe de tip *Spline*, construite prin punctele succesive obţinute din rularea mecanismului. Coordonatele acestor puncte se pot extrage sub forma unui fişier text sau orice program de prelucrat foi de calcul, precum *OpenOffice*, figura 9. 18.

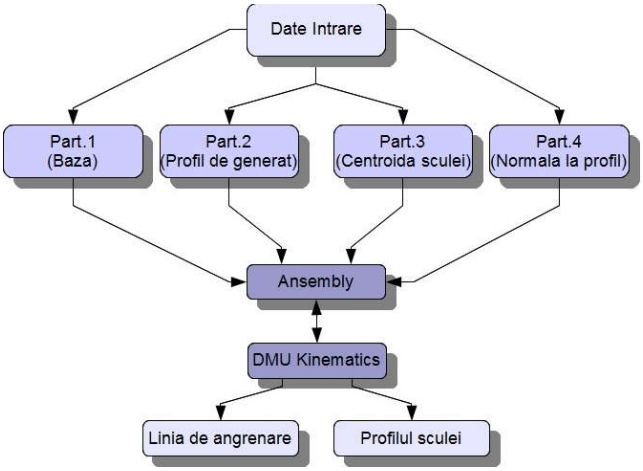


Figura 9. 18. Algoritm de generare în mediul grafic de proiectare CATIA

Soluția propusă are avantajul că utilizează facilitățile unui produs soft deosebit de versatil, care poate oferi o deosebită rigurozitate a rezultatelor numerice.

Totodată, fiind o metodă grafică, erorile grosolane, datorate în primul rând formării curbelor de trecere, ce pot fi considerate eronat porțiuni ale profilului, sunt uşor de sesizat şi, ulterior, de eliminat din analiză.

Se consideră trei tipuri de profiluri ale pieselor, pentru care se proiectează tot atâtea tipuri de mecanisme virtuale de generare în mediul CATIA (M.G.M.C.) diferite. Aceste mecanisme sunt trecute în tabelul 9. 6.

Prin combinarea tuturor acestor tipuri de mecanisme se pot studia profiluri compuse şi complexe, după cum se va observa în continuare. Pentru cazul general al oricărui profil mecanismul este alcătuit din elementele următoare:

- Bază;
- Piesă;
- Tachet;
- Sculă.

Tabelul 9. 6. Tipuri de M.G.M.C.

Tip M.G.M.C.	***Tip profil piesă***
M.G.M.C. pentru segment de dreaptă	Profiluri formate din segmente rectilinii
M.G.M.C. pentru arc de cerc	Profiluri formate din arce de cerc, tangente sau nu în punctele de contact
M.G.M.C. pentru curbă de tip Spline	Profiluri formate din curbe, date prin puncte sau cunoscute prin ecuații (evolvente, cicloide, etc.)

Metoda analitică

Principial, problematica determinării profilului sculei cremalieră reciproc înfăşurătoare unui vârtej ordonat de profiluri, asociat unei centroide circulare, presupune respectarea cinematicii procesului generării, figura 9. 19.

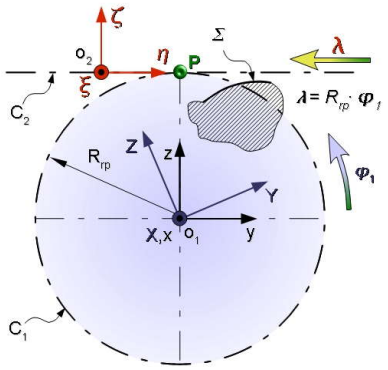


Figura 9. 19. Cuplul de centroide în rulare; Cinematica generării

Cele două centroide în rulare – C_1, asociată vârtejului ordonat de profiluri (suprafețe) de generat și C_2 – rectilinie, asociată spațiului sculei cremalieră, se află în mișcare de rulare astfel încât este, în permanență, respectată condiția

$$\lambda = R_{rp} \cdot \varphi_1 \tag{9.89}$$

în care:

λ este viteza liniară în mișcarea de translație a centroidei C_2 ;

$R_{rp} \cdot \varphi_1$ mărimea vitezei în punctul O_1 – polul angrenării, de pe centroida C_1, aflată în mișcare de rotație în jurul axei z;

φ_1 parametru unghiular variabil.

În mișcarea de rotație a centroidei C1, în mod curent, mișcările de translație în lungul centroidei C2 și rotația în jurul axei Z, sunt uniforme.

Se definesc sistemele de referință:

xyy este sistemul de referință fix, având axa x suprapusă axei de rotație a centroidei a centroidei C_1;

XYZ, ξηζ- sisteme de referință mobile.

Cinematica principială a procesului de rulare a celor două centroide, C_1 și C_2, tangente în punctul O_1 – polul angrenării – presupune ca vitezele punctelor aparținând celor două centroide, vremelnic aflate în punctul O_1, să fie egale.

Astfel, mișcarea absolută a sistemului ξηζ, solidar centroidei C_2, este descrisă de transformarea,

$$x = \xi + a \tag{9.90}$$

în care:

$$\xi = \begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix}; \; x = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{9.91}$$

reprezintă matricea punctului curent din spațiul ξηζ, respectiv xyz;

$$a = \begin{pmatrix} 0 \\ -\lambda \\ -R_{rp} \end{pmatrix} \tag{9.92}$$

este matricea formată cu coordonatele punctului O_1, în sistemul de referință fix;

λ- viteza instantanee în mișcarea de translație a centroidei C_1;

R_{rp} - mărimea razei centroidei circulare C_1 (raza de rulare).

De asemenea, mișcarea de rotație a centroidei C_1 este descrisă de transformarea:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \omega_1^T(\varphi_1) \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{9.93}$$

în care: $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ este matricea punctului curent al spațiului XYZ;

$$\omega_1(\varphi_1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi_1) & \sin(\varphi_1) \\ 0 & -\sin(\varphi_1) & \cos(\varphi_1) \end{pmatrix} \tag{9.94}$$

este matricea transformării de rotație, în jurul axei X, de unghi φ_1 (rotație în sens trigonometric).

Ansamblul de ecuații (9.90) şi (9.93), cu respectarea condiției de rulare (9.89), determină mişcarea relativă,

$$\begin{pmatrix} \xi \\ \eta \\ \zeta \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi_1) & \sin(\varphi_1) \\ 0 & -\sin(\varphi_1) & \cos(\varphi_1) \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} - \begin{pmatrix} 0 \\ -\lambda \\ -R_{rp} \end{pmatrix} \tag{9.95}$$

în decursul căreia profilul Σ, aparținând vârtejului ordonat de profiluri, asociat centroidei C_1, în forma,

$$\Sigma = \begin{pmatrix} 0 \\ Y(u) \\ Z(u) \end{pmatrix} \tag{9.96}$$

cu u – parametru variabil, descrie o familie de profiluri în spațiul cremalierei,

$$(\Sigma)_{\varphi_1} \begin{cases} \xi(u,\varphi_1) = 0; \\ \eta(u,\varphi_1) = Y(u)\cos(\varphi_1) - Z(u)\sin(\varphi_1) + R_{rp} \cdot \varphi_1; \\ \zeta(u,\varphi_1) = Y(u)\sin(\varphi_1) + Z(u)\cos(\varphi_1) - R_{rp}. \end{cases} \tag{9.97}$$

Se asociază familiei de profiluri (9.97) o condiție de înfăşurare definită în formă analitică, în baza uneia dintre teoremele fundamentale sau metodelor complementare.

Dacă se acceptă condiția specifică metodei traiectoriilor plane de generare, în forma

$$\eta'_u \cdot \xi'_{\varphi_1} - \eta'_{\varphi_1} \cdot \xi'_u = 0 \tag{9.98}$$

în care $\eta'_u, \xi'_{\varphi_1}, \eta'_{\varphi_1}, \xi'_u$ reprezintă derivatele parțiale, calculate din (9.97), atunci, ansamblul de ecuații (9.97) şi (9.98) reprezintă profilul sculei cremalieră.

Pentru $\varphi_1 = const.$, se determină, la un moment dat, coordonatele punctului de contact între scula cremalieră şi semifabricat, în sistemul de referință al sculei.

Deoarece condiția (9.98) reprezintă, principial, o legătură între parametrii u şi φ_1,

$$u = u(\varphi_1) \tag{9.99}$$

atunci, familia de profiluri (9.97) se transformă în forma:

$$\Sigma_{(\varphi_1)} \begin{cases} \xi = 0; \\ \eta = \eta(\varphi_1); \\ \zeta = \zeta(\varphi_1), \end{cases} \tag{9.100}$$

reprezentând profilul sculei cremalieră.

Se propune, în scopul validării metodei CAD, aşa cum a fost concepută aceasta, realizarea unei comparații, pentru diferite exemple de aplicare, pentru profiluri simple: segment de dreaptă, arc de cerc, profil evolventic ale semifabricatului, şi forma (numerică) a profilului cremalierei generatoare.

Generarea profilurilor rectilinii (arbori poligonali)

Pentru a obține profilul sculei pentru o varietate de arbori poligonali, se procedează în două etape. Prima etapa este alegerea lungimii laturii şi a diametrului cercului de rulare, ca date de intrare.

Acest lucru se poate face introducând într-un fişier text sau *Excel* valorile, urmând ca, în programul CATIA, să se modifice automat întregul mecanism, după noile valori introduse. Tot în aceasta prima etapă are loc şi crearea mecanismului, rularea acestuia şi obținerea profilului sculei. Exportul coordonatelor punctelor de pe profil se face fie în format text fie în format excel.

A doua etapă constă în refacerea parțială a cuplelor mecanismului, rularea acestuia pentru a obține noul profil al sculei, pentru valorile modificate anterior, şi exportul într-un nou fişier a punctelor de pe profil.

În figura 9. 20, este prezentat un arbore poligonal şi sistemele de referință față de care este raportat.

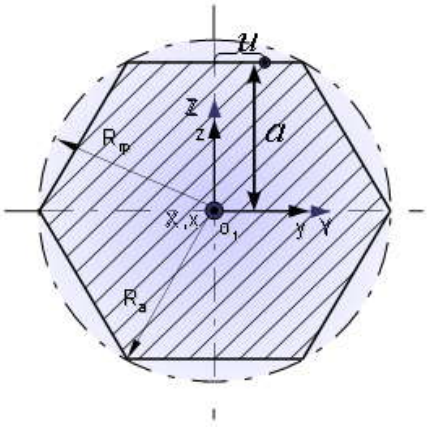


Figura 9. 20. Secțiunea transversală a arborelui hexagonal

M.G.M.C. pentru segment de dreaptă

Mecanismul virtual specific acestui caz (profil reprezentat de un segment de dreaptă) este prezentat în figura 9. 21.

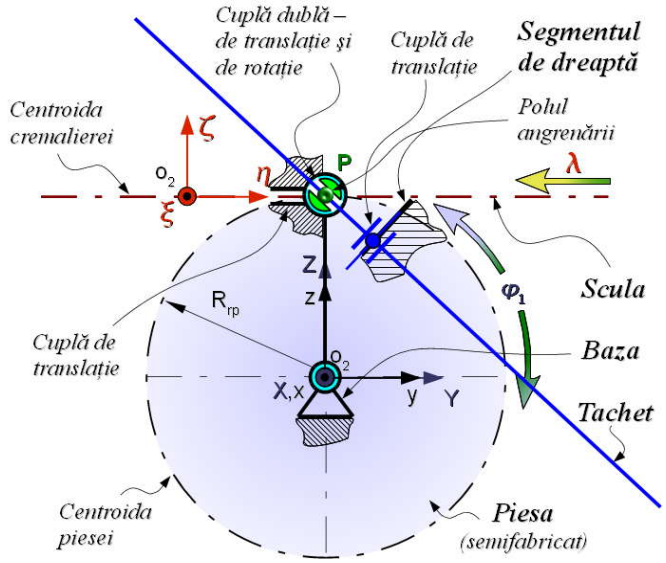


Figura 9. 21. M.G.M.C. pentru segment de dreaptă şi scula cremalieră

Elementul *Scula* este o dreaptă, ce reprezintă centroida asociată cremalierei, şi care rulează fără alunecare cu cercul reprezentând centroida piesei.

Elementul *Baza* este elementul fix ce permite, prin cuplele de rotaţie din origine şi cea din polul angrenării, rotaţia piesei şi respectiv rotaţia tachetului. Elementul *Tachet* are o legătură cu *Baza* şi alta cu *Piesa*, care este şi cea specifică pentru profilul segment de dreaptă. Această cuplă este cea de translaţie (de tip prismatic) a elementului cremalieră in lungul unui ghidaj al elementului fix, *Baza*.

În figura 9. 22, este reprezentat fişierul de tip *Ansembly* cu M.G.M.C. specific, aplicat pentru arborele hexagonal, care prin comanda *Simulation* realizează rularea celor două centroide, a semifabricatului şi a sculei cremalieră.

Tabelul 9. 7. Cuplele folosite în mediul DMU Kinematics

Nr. Crt.	***Tip cuplă (Joint)***	***Elementele mecanismului***	
1	Fixă (*Fix*)	Baza	-
2	De rotaţie (*Revolute*)	Axa (*Baza*)	Axa (*Piesa*)
3	De Translaţie (*Prismatic*)	Ghidaj Cremaliera (*Baza*)	Dreapta de rulare (*Cremaliera*)
4	Cremaliera *(Rack = Prismatic + Revolute)*	GhidajDreaptaRulare, AxaArbore şi Plan YZ (*Baza*)	DreaptaRulare, AxaArbore şi Plan YZ (*Cremaliera*)
5	De Translaţie (*Prismatic*)	Tangenta (*Tachet*)	Latura profilului L (*Piesa*)
6	Punct pe curba (*PointCurve*)	PolulAngrenarii (*Baza*)	Normala (*Tachet*)
7	Punct pe curba (*PointCurve*)	PunctContact (*Tachet*)	Latura profilului L (*Piesa*)

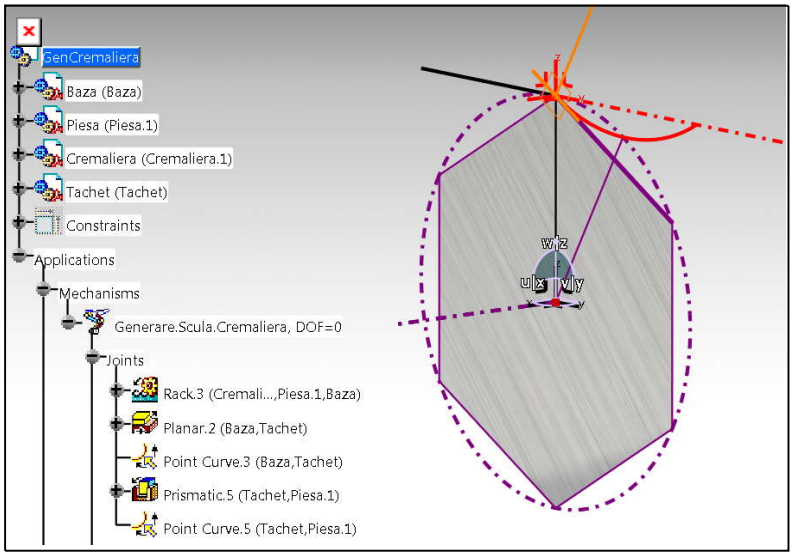


Figura 9. 22. Fişierul de tip Ansembly cu cuplele cinematice şi profilul sculei

Rularea mecanismului – aflarea profilului sculei

În figura 9. 23, pentru un număr de 500 de poziţii intermediare, este trasat profilul sculei cremalieră, pentru arborele hexagonal cu latura de 50 mm (raza de rulare, Rrp=50 mm).

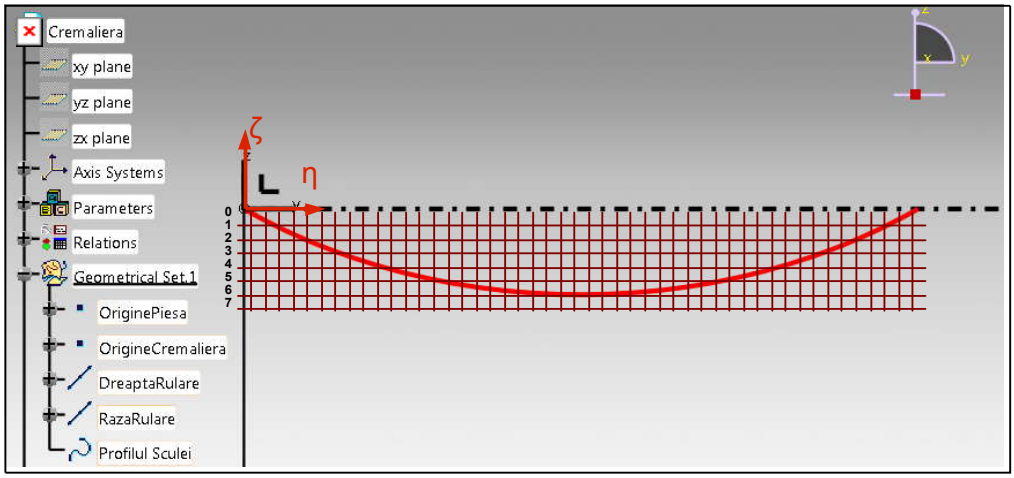


Figura 9. 23. Profilul Sculei în sistemul de axe al cremalierei ξηζ

Profilul sculei este obținut prin lansarea comenzii *Trace* şi alegerea ca element de trasare a profilului punctul *PunctContact*, de pe *Tachet*, ce va trasa, în sistemul de referință al cremalierei, profilul sculei, care este salvat automat într-un alt fişier denumit *SculaHexagon.part* şi introdus, apoi, în fişierul ansamblu.

Acest profil este o curbă de tip *Spline* ce trece prin 500 puncte şi care va face parte din fişierul *Cremaliera*, sau va fi rigidizată de acesta cu ajutorul cuplei *Rigid*.

Generarea profilurilor în arc de cerc (roata de lanț)

Profilarea sculei cremalieră pentru un profil compus din arce de cerc, cum este cazul roții de lanț, figura 9. 24, impune crearea în mediul *DMU Kinematics* a unui mecanism virtual specific pentru generarea prin înfăşurare a unui arc de cerc, vezi figura 9. 25.

Pentru determinarea profilului sculei, care generează flancul unui dinte al unei roți de lanț, se procedează în acelaşi mod ca şi în cazurile anterior prezentate. Fişierul denumit *Piesa* va conține două schițe cu profilurile flancului, schițe reprezentate de două arce de cerc, tangente între ele, figura 9. 25.

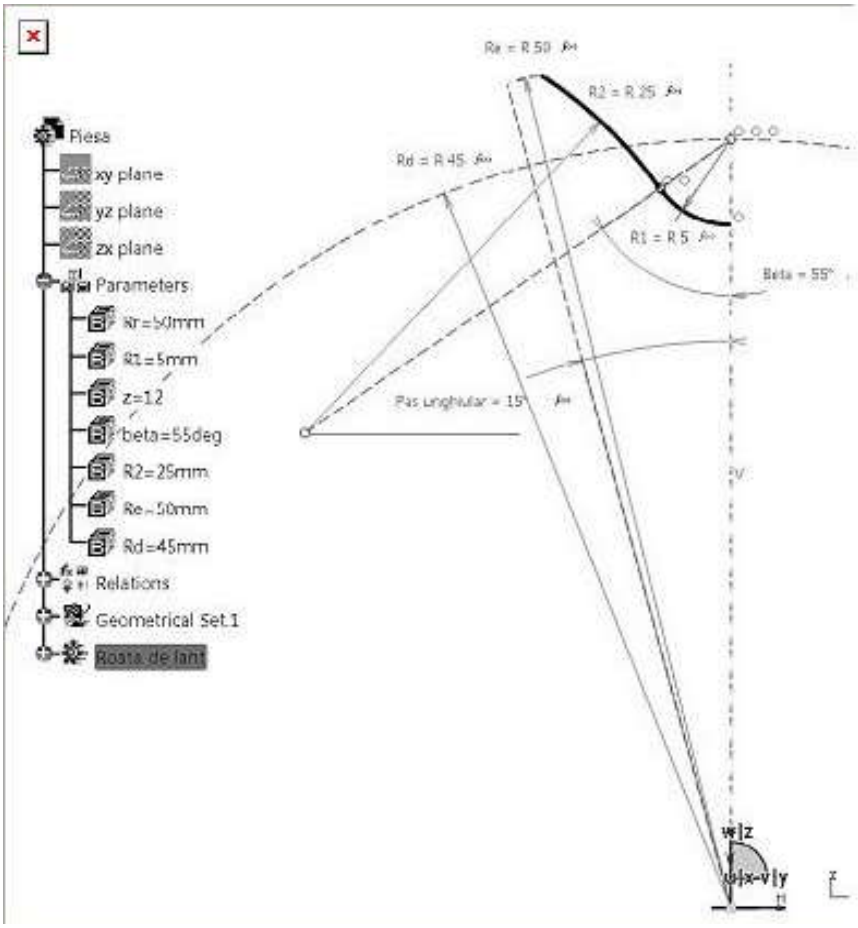


Figura 9. 24. Geometria profilului unui flanc al dintelui roții de lanț

Mecanism virtual pentru generare în mediul CATIA a arcelor de cerc

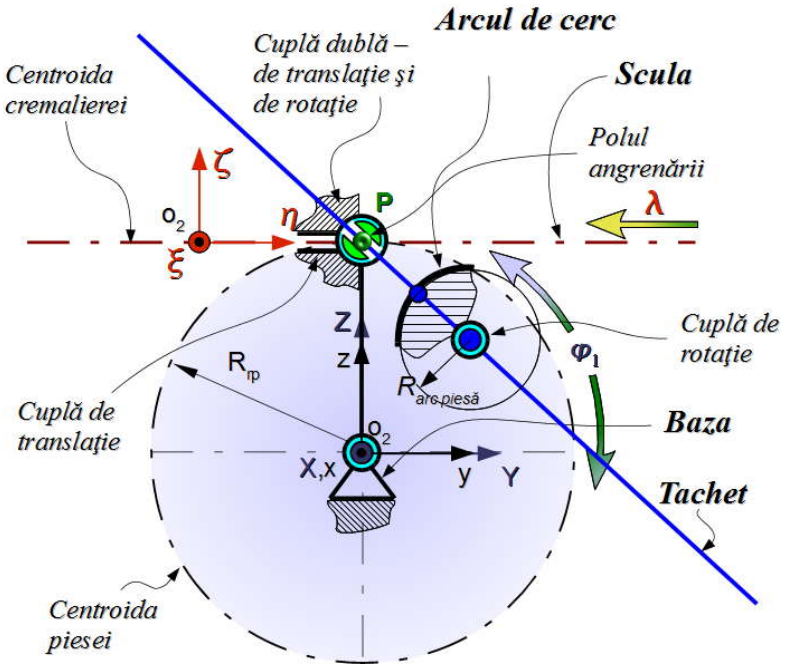


Figura 9. 25. M.G.M.C. pentru un arc de cerc

Particularitatea, față de mecanismul virtual pentru generare în mediul CATIA (M.G.M.C., pentru segmente de dreaptă), constă în faptul că este înlocuită cupla de translație dintre *Tachet* şi *Piesa* (profil) cu o cuplă de rotație în jurul unei axe, care trece prin centrul arcului reprezentând profilul.

Celelalte elemente şi comenzi rămân neschimbate.

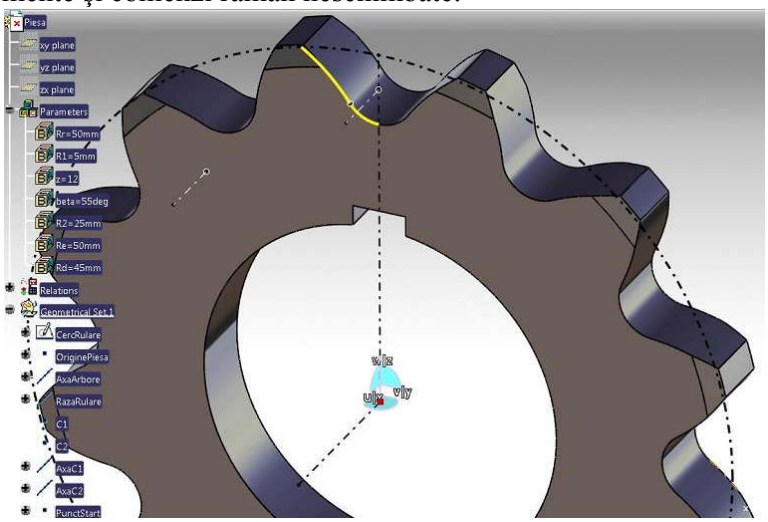


Figura 9. 26. Roata de lanț – ansamblu – fişierul Piesa

După crearea fişierului *Piesa* se trece în mediul de lucru *DMU Kinematics* pentru a crea cuplele mecanismului specifice profilurilor curbe. În acest, scop se înlocuieşte cupla *Prismatic* de la profilurile rectilinii, cu cuplele *Revolute* pentru axele din centrele arcelor de cerc ce reprezintă profilul.

Acest tip de cuplă permite rotirea elementului *Tachet*, prin intermediul axelor din centrele arcelor de cerc (*AxaC1* şi *AxaC2*), în jurul axelor similare din fişierul *Piesa*, în timp ce cupla *PointCurve*, dintre *PunctContact,* din fişierul *Tachet*, şi *ProfilPiesă,* din fişierul *Piesa*, rămâne ca element conducător.

În mediul DMU Kinematics, se vor realiza două mecanisme, vezi figura 9. 27, pentru fiecare arc de cerc al profilului compus al roții de lanț, vezi şi tabelul 9. 8.

Tabelul 9. 8. Cuplele cinematice pentru profilul unei roți de lanț

Nr. Crt.	***Tip cuplă (Joint)***	***Elementele mecanismului***	
1	Rigidă (*Fix*)	Baza	-
2	De rotație (*Revolute*)	Axa (*Baza*)	Axa (*Piesa*)
3	De Translație (*Prismatic*)	Ghidaj Cremaliera (*Baza*)	Dreapta de rulare (*Cremaliera*)
4	Rostogolire fără alunecare *(RollCurve)*	Cerc de rulare (*Piesa*)	Cerc de rulare (*Cremalieră*)
5	De rotație (*Revolute*)	AxaC1 şi AxaC2 (*Baza*)	AxaC1 şi AxaC2 (*Piesa*)
6	Punct pe curba (*PointCurve*)	PolulAngrenarii (*Baza*)	Normala (*Tachet*)
7	Punct pe curba (*PointCurve*)	PunctContact (*Tachet*)	Latura profilului L (*Piesa*)

După crearea mecanismelor, a simularilor şi după înregistrarea cinematicii acestora, cu comanda *Trace* se trasează pe rând, profilul sculei.

În figura 9. 27 şi tabelul 9. 9, este prezentat profilul sculei cremalieră şi coordonatele acestuia, pentru roata de lanţ având caracteristicile: $R_r=R_e=55$mm; $z=12$ dinţi; $R_2=25$mm; $R_1=5$mm; $\beta=55^\circ$ şi $R_d=45$mm.

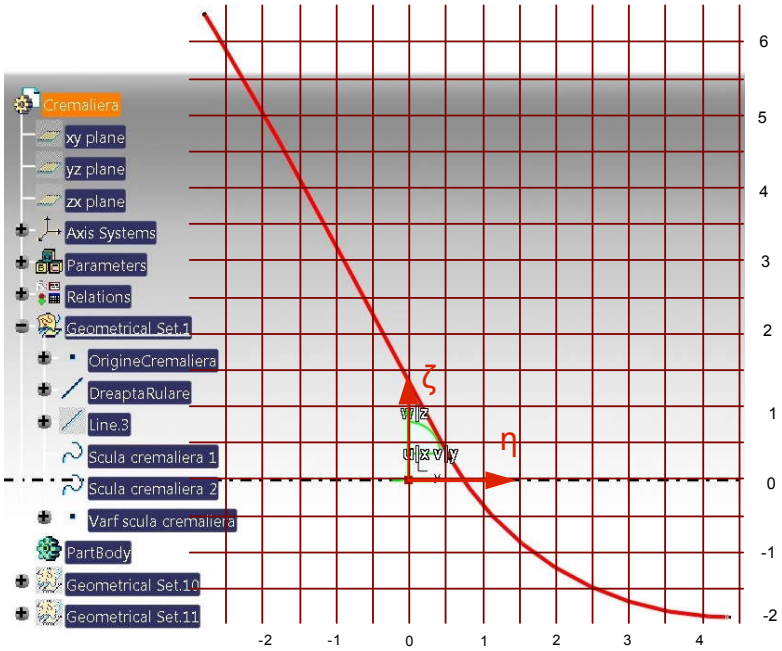


Figura 9. 27. Profilul sculei pentru roata de lanţ

Tabelul 9. 9. Coordonatele punctelor de pe profilul sculei-cremalieră

Nr. Crt.	*η[mm]*	*ζ[mm]*	*Nr. Crt.*	*η[mm]*	*ζ[mm]*	*Nr. Crt.*	*η[mm]*	*ζ[mm]*
1	-11.0774	0.0000	***71***	-5.6660	-5.8441	***36***	-2.2752	-9.3640
8	-10.4659	-0.6308	***78***	-5.0435	-6.5328	***43***	-2.0707	-9.4823
15	-9.9822	-1.1376	***85***	-4.3956	-7.2481	***50***	-1.8521	-9.5923
22	-9.4899	-1.6599	***92***	-3.7201	-7.9913	***57***	-1.6214	-9.6919
29	-8.9875	-2.1985	***100***	-3.0146	-8.7635	***64***	-1.3803	-9.7793
36	-8.4734	-2.7547	***1***	-3.0146	-8.7635	***71***	-1.1304	-9.8535
43	-7.9460	-3.3298	***8***	-2.9125	-8.8690	***78***	-0.8734	-9.9132
50	-7.4034	-3.9249	***15***	-2.7848	-8.9878	***85***	-0.6111	-9.9578
57	-6.8439	-4.5415	***22***	-2.6344	-9.1131	***92***	-0.3450	-9.9866
64	-6.2655	-5.1807	***29***	-2.4638	-9.2399	***100***	0.0000	-10.0000

Verificarea calităţii metodei de profilare a sculelor în mediul CATIA

Este propusă verificarea metodei de profilare a sculelor în mediul de proiectare CATIA, prin compararea rezultatelor obţinute prin această metodă cu rezultatele obţinute printr-una dintre metodele analitice consacrate: metoda Gohman; metoda normalelor sau metoda distanţei minime.

Pentru acest lucru, a fost considerat profilul format din segmente de dreaptă prezentat în secţiunea "*Generarea profilurilor rectilinii (arbori poligonali)*".

Forma analitică a profilului de generat este descrisă de ecuaţiile:

$$\Sigma \begin{vmatrix} X = 0; \\ Y = u; \\ Z = a, \end{vmatrix} \qquad (9.101)$$

cu *u* parametru variabil, măsurat în lungul profilului şi *a* valoare constantă dependentă de forma profilului, vezi figura 9. 28.

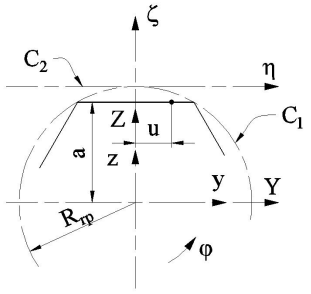


Figura 9. 28. Profil de tip segment de dreaptă

Familia de profiluri Σ are forma:

$$(\Sigma)_{\varphi}\begin{vmatrix}\xi = 0;\\ \eta = u\cos\varphi - a\sin\varphi + R_{rp}\varphi;\\ \zeta = u\sin\varphi + a\cos\varphi - R_{rp},\end{vmatrix} \tag{9.102}$$

cu φ parametru variabil.

Condiția specifică de înfăşurare, conform metodei traiectoriilor plane de generare este:

$$\frac{\cos\varphi}{\sin\varphi} = \frac{-u\sin\varphi - a\cos\varphi + R_{rp}}{u\cos\varphi - a\sin\varphi} \tag{9.103}$$

sau

$$u = R_{rp}\sin\varphi \tag{9.104}$$

Ansamblul de ecuații (9.102) şi (9.104), pentru u variabil între limitele $u_{\min} = -0.5\cdot R_{rp}$; $u_{\max} = 0.5\cdot R_{rp}$, pentru arborele de secțiune hexagonală, reprezintă profilul sculei cremalieră reciproc înfăşurătoare cu profilul de generat.

În tabelul 9. 10, sunt prezentate coordonatele profilului determinate prin metoda analitică menționată (metoda traiectoriilor plane de generare).

În figura figura 9. 29, sunt prezentate formele profilurilor sculei şi erorile între acestea la profilarea prin cele două metode.

NOTA: Pentru a putea face această comparație s-a utilizat transformarea de coordonate:

$$\begin{aligned}\zeta_{CATIA} &= -\xi_{JAVA};\\ \eta_{CATIA} &= \eta_{JAVA} + 26.1799.\end{aligned} \tag{9.105}$$

Tabelul 9. 10. Coordonatele punctelor de pe profilul sculei—metoda traiectoriilor plane de generare, în programul Java

Nr. crt.	ξ [mm]	η [mm]	Nr. crt.	ξ [mm]	η [mm]	Nr. crt.	ξ [mm]	η [mm]
1	1.40E-06	-26.1799	**248**	6.698025	-0.28286	**496**	0.199296	25.83104
2	0.05004	-26.093	**249**	6.698477	-0.1695	**497**	0.149679	25.9186
3	0.099941	-26.0059	**250**	6.698703	-0.05562	**498**	0.099923	26.00594
4	0.149712	-25.9185	**251**	6.698701	0.057729	**499**	0.050028	26.09306
5	0.19934	-25.831	**252**	6.698471	0.171609	**500**	8.88E-07	26.17994
⋮	⋮	⋮	⋮	⋮	⋮	⋮		

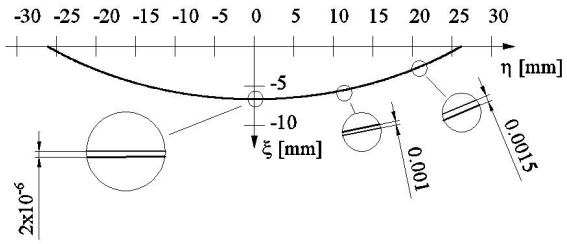


Figura 9. 29. Erorile obținute la profilarea sculei prin metoda cinematică față de profilarea sculei prin metoda traiectoriilor plane de generare

Cele prezentate mai sus demonstrează capacitatea metodei cinematice de a fi utilizată pentru profilarea sculelor care generează prin înfăşurare, prin metoda rulării.

Mecanismul *Tachet* utilizat în rezolvarea acestui tip de probleme are un grad ridicat de generalitate a utilizării.

Precizia de profilare a sculei cremalieră este comparabilă pentru cele două metode.

9.2.2. Produse soft pentru profilarea sculei cremalieră

Produse soft bazate pe metoda cinematică în mediul grafic de proiectare CATIA

Pentru a uşura mult procesul de generare a profilurilor piesei, a mecanismelor, a cuplelor elementelor specifice fiecărui caz în parte, a profilurilor sculelor şi liniilor de angrenare, s-a elaborat un soft specializat în mediul de programare *Visual Basic for Aplications*, integrat în CATIA, prin care, cu ajutorul unei interfețe grafice, se pot configura în mod automat aceste particularități, prin introducerea datelor de intrare şi, în final, extragerea în fişiere de tip text, a punctelor de pe aceste profiluri.

În figura 9. 30, este prezentată interfața programului pentru cazul arborelui poligonal. Datele de intrare în acest caz sunt: numărul de laturi *NrL*, raza cercului circumscris *Re*, raza cercului înscris *Ri*, lungimea laturii *L*.

Raza de rulare a semifabricatului poate varia ca mărime între raza cercului circumscris hexagonului şi raza cercului înscris în hexagon. Programul permite alegerea uneia dintre aceste valori.

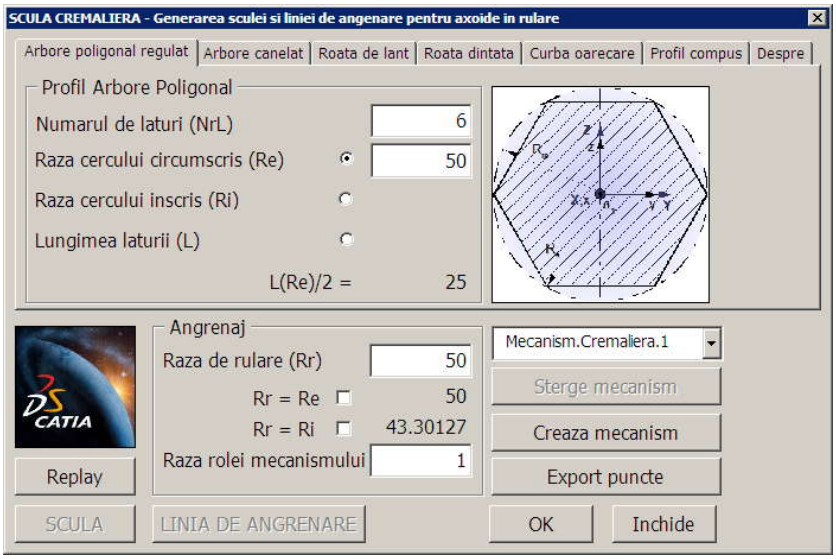


Figura 9. 30. Interfața aplicației VBA pentru arbore poligonal

Pentru arborele canelat, pagina aplicației arată ca în figura 9. 31, în care datele de intrare sunt: numărul de caneluri *NrC*, grosimea canelurii *b*, raza interioară r_i, raza exterioară R_e.

Raza de rulare a semifabricatului poate varia între limitele: raza exterioară R_e şi cea minimă de rulare acceptată,

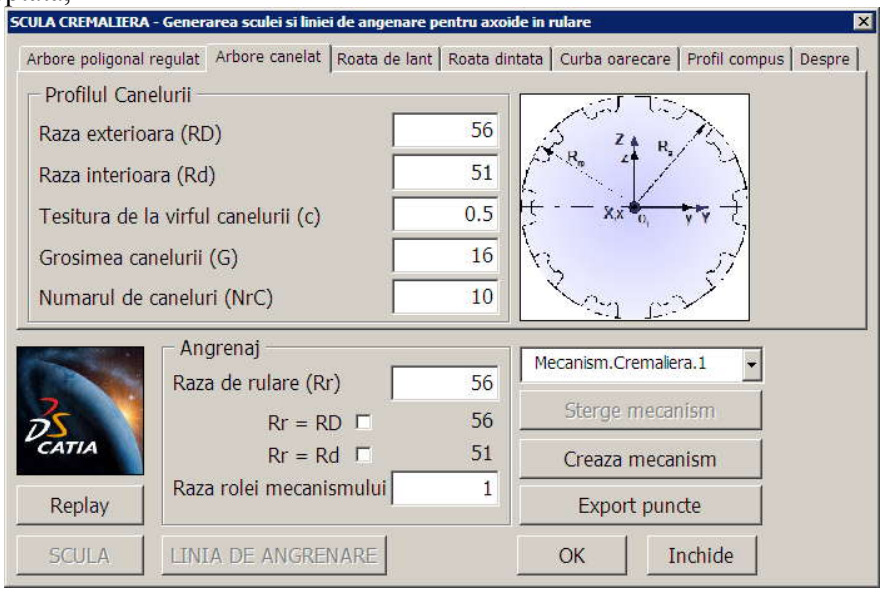


Figura 9. 31. Interfața aplicației VBA pentru arbore canelat

Datele de intrare pentru profilul flancului unui dinte de roată de lanț, sunt: pasul roții *P*, numărul de dinți *z*, raza rolei de lanț r_1, unghiul de tangență β, unghiul γ, figura 9. 32.

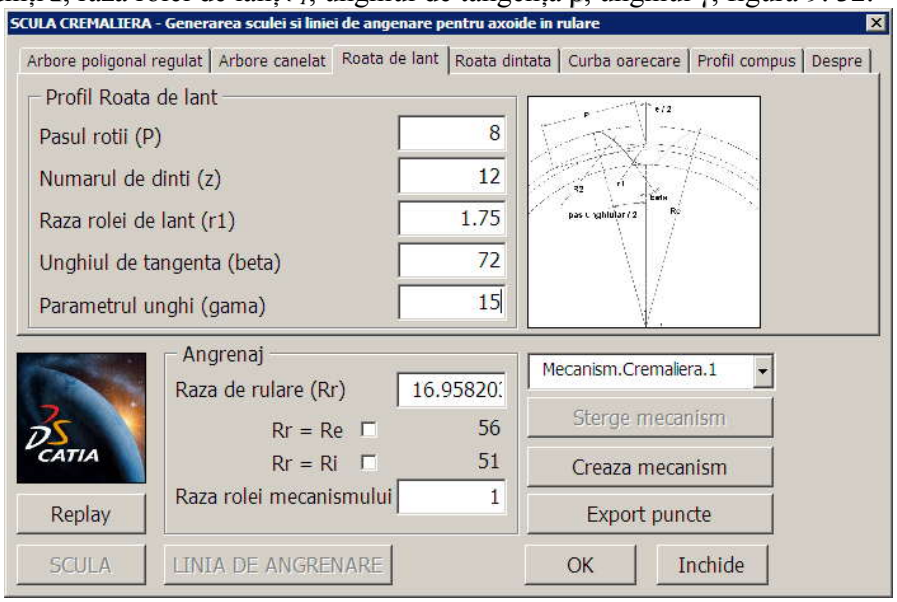


Figura 9. 32. Interfața aplicației VBA pentru roata de lanț

Profilul evolventic este construit în CATIA folosind formulele evolventei pentru fiecare componentă a coordonatelor, pe *y* şi respectiv pe *z*, iar datele de intrare determină acest profil evolventic cât şi forma întregii roți.

Aceste date de intrare sunt introduse în pagina corespunzătoare din aplicație, vezi figura 9. 33.

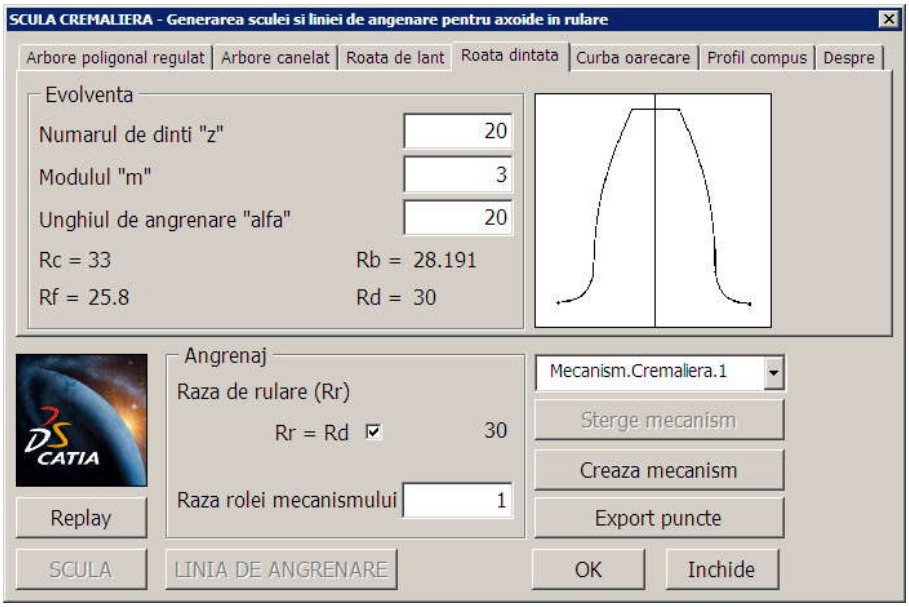


Figura 9. 33. Interfața aplicației VBA pentru roata dințată

În cazul profilului compus din pagina aferentă aplicației, vezi figura 9. 34, sunt introduse aceste profiluri cu ajutorul comenzii *Adaugă profil*, oricâte ca număr, prin introducerea inițială a punctului de început, apoi succesiv, a următoarelor puncte şi a tipului de profil dorit.

Pentru editarea acestor profiluri se apelează comanda *Editare profil*, ceea ce înseamnă că acestea pot fi modificate uşor şi chiar şterse prin apelarea comenzii *Sterge profil*.

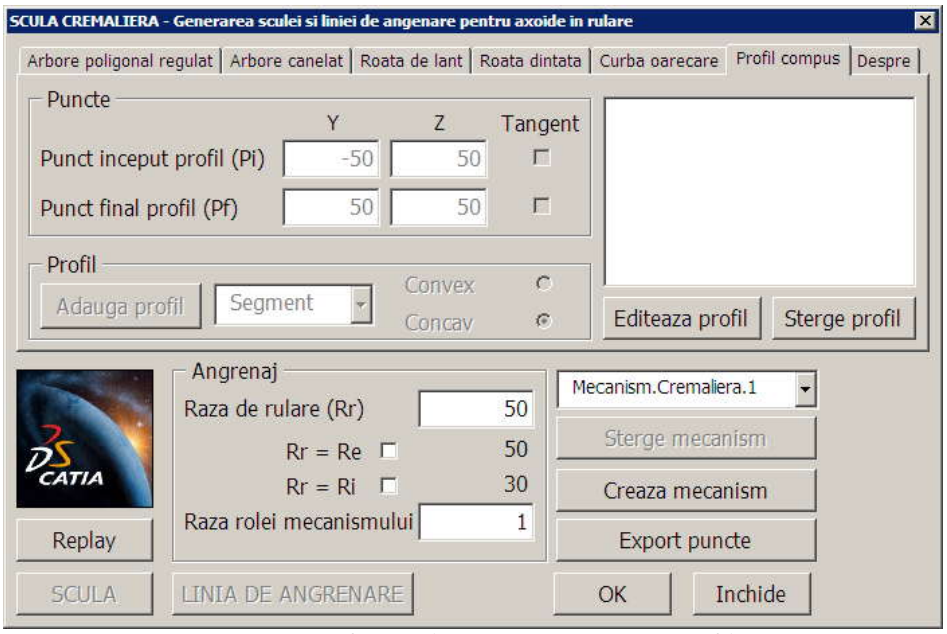


Figura 9. 34. Interfața aplicației VBA pentru profil compus

În toate aceste cazuri, mecanismele şi cuplele aferente sunt create automat apelând comanda *Crează mecanism*, iar dacă se doreşte ştergerea unei configurații, se apelează comanda *Şterge mecanism*.

În toate cazurile discutate, raza de rulare poate fi aleasă pentru fiecare caz în parte, dar asigurându-se respectarea teoremei fundamentale a angrenării, teorema *Willis*.

Scula cremaliera

Aplicația, realizată în cadrul proiectului, permite calcularea profilului sculei cremalieră pornind de la profilul piesei de generat. Se defineşte un ansamblu de profiluri elementare drept profil transversal al piesei de generat. Aplicația permite calculul profilului sculei cremalieră ce generează prin înfăşurare piesa dorită. De asemenea, se calculează un profil aproximativ al sculei cremalieră, utilizînd polinoame Bezier. Eroarea de aproximare este determinată de program.

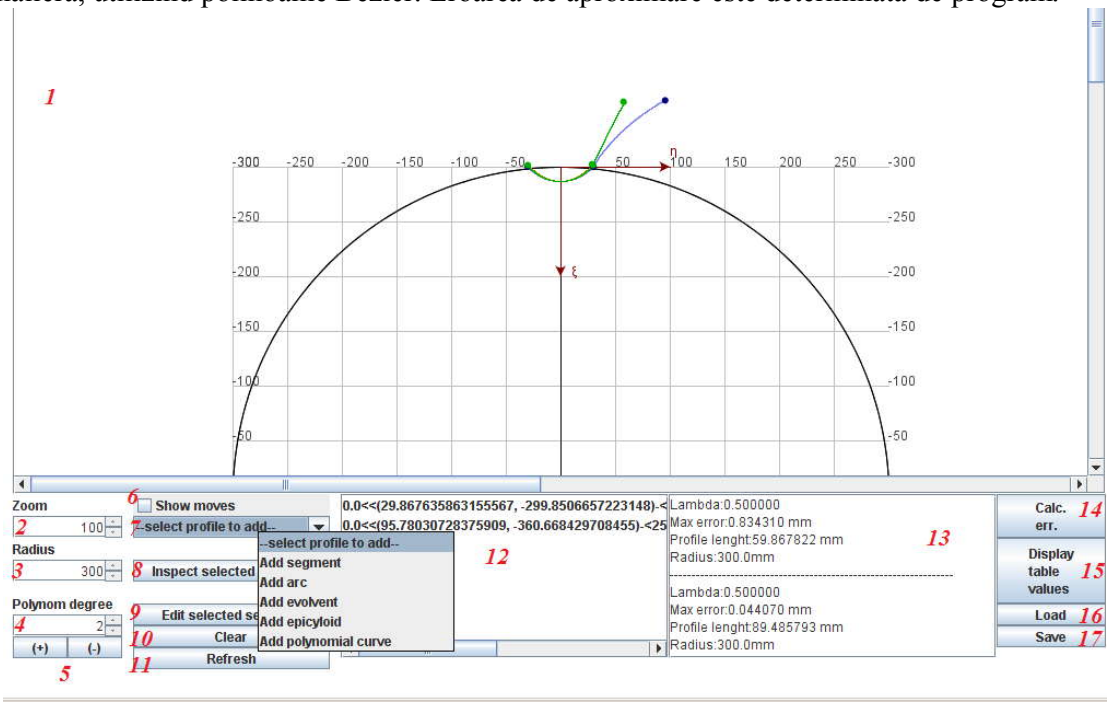


Figura 9. 35. Applet- Scula cremalieră

Principalele elemente vizuale şi funcționale ale aplicației (figura 9. 35):

1 – zona în care sunt afişate profilurile piesei şi ale sculei;

2 – controlează nivelul de *zoom*;

3 – raza de bază a semifabricatului;

4 – gradul polinomului de aproximare;

5 – creşte/scade nivelul de zoom;

6 – check-box pentru afişarea profilurilor în mişcare;

7 - selecția tipului profilului elementar ce va fi adaugat în ansamblul de profiluri:

- segment de dreapta (definit prin punctele de capat);
- arc de cerc (definit prin punctele de capat şi raza cercului din care face parte);
- evolventa (definită prin punctul de start, raza de baza, raza interioară şi exterioară);
- epicicloidă (definită prin punctul de start, raza cercului de rulare (r), raza cercului fix (R), şi unghiul de variație (alpha));
- curbă polinomială (aproximarea polinomială a unui set de puncte măsurate);

8 – după selectarea unui profil din lista (12), vezi figura 9. 36 se alege o valoare pentru lambda între 0 şi 1, coordonatele corespunzătoare sunt afişate;

9 – după selectarea unui profil din lista (12), se pot modifica parametrii de definiție ai profilului respectiv;

10 – şterge lista de profiluri (12) ;

11 – redesenează zona (1);

12 – afişează lista de profiluri elementare ce compun ansamblul;

13 – zona de afişare de mesaje

14 – calculează eroarea de aproximare dintre profilul teoretic al sculei cremalieră şi profilul obținut pe baza polinoamelor Bezier de aproximare;

15 – afişează rezultatele comparative obținute prin (12) (profilul teoretic şi profilul aproximat al sculei) în formă tabelară;

16 – încarcă ansamblul de profiluri dintr-un fişier salvat anterior;

15 – salvează ansamblul de profiluri într-un fişier.

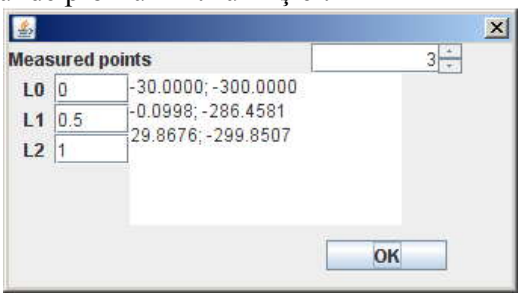


Figura 9. 36. Adăugare segment de dreaptă

În continuare, sunt descrise în detaliu ferestrele de configurare ale profilurilor elementare.

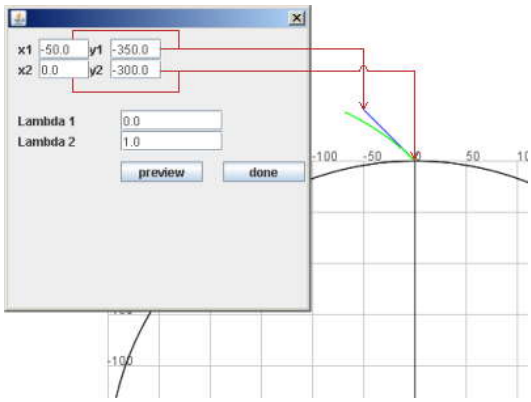


Figura 9. 37. Adăugare segment de dreaptă

Adăugarea unui segment de dreaptă la lista de fracțiuni ale profilului piesei, segment determinat de punctele (x1, y1) şi (x2, y2) (segmentul este marcat cu albastru în figură). Lambda1 şi Lambda2 reprezintă prelungirile virtuale ale segmentului de dreaptă (când Lambda1 < 0 sau Lambda2 > 1). Butonul **"Preview"** adaugă segmentul la profil, redesenează figura, dar nu închide fereastra de editare. Butonul **"Done"** închide fereastra de editare (şi adaugă segmentul la profil dacă nu a fost deja adăugat prin butonul "Preview")

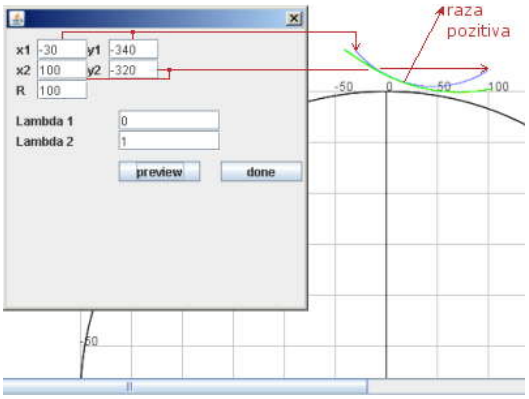


Figura 9. 38. Adăugare arc de cerc

Adăugarea un arc de cerc la lista de fracțiuni ale profilului piesei, cu capetele determinate de punctele (x1, y1) şi (x2, y2) şi raza R.(arcul este marcat cu albastru in figura). Centrul cercului se află la distanță egală, R, pe mediatoarea segmentului x1,y1 - x2,y2 față de punctele x1,y1 şi x2,y2. Schimbarea semnului parametrului R schimba poziția centrului cercului din care face parte arcul, față de segmentul x1,y1 - x2,y2.

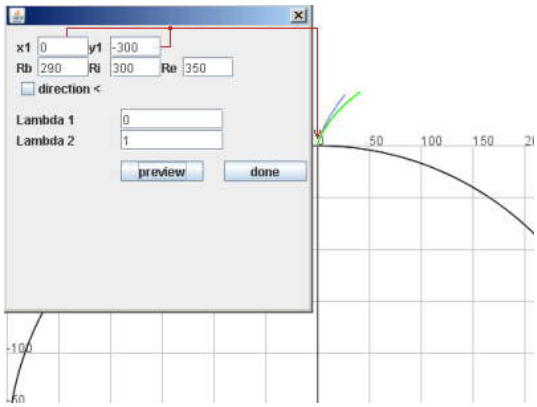


Figura 9. 39. Adăugare evolventă

Adăugarea unui arc deevolventă la lista de fracțiuni ale profilului piesei. Cercul de bază al evolventei are raza Rb. Aceasta este trasată de la intersecția cu cercul de rază Ri până la intersecția cu cercul de rază Re. (evolventa este marcată cu albastru în figură). Capătul inițial al evolventei este poziționat în punctul (x1,y1).
Evolventa este trasată în sensul acelor de ceasornic, iar dacă este bifată căsuța **"direction <"** ,atunci evolventa va fi trasată în sens trigonometric.

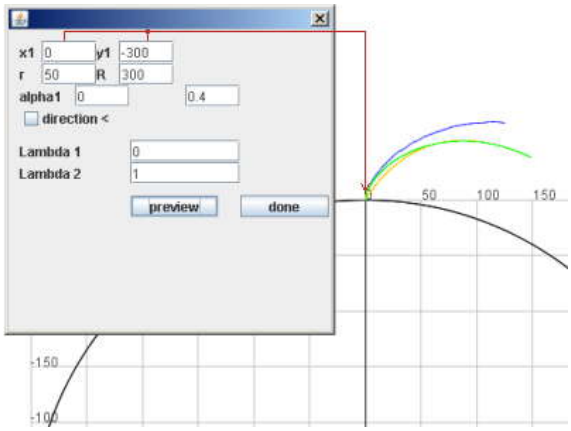


Figura 9. 40. Adăugare epicicloidă

*Adăugarea unui arc de epicicloidă la lista de fracțiuni ale profilului p*iesei. Ruleta are raza r iar cercul fix are raza R. Parametrul alfa are o valoare de $\frac{2\pi r}{R}$ pentru o rotație completă a epiciclului pe cercul fix. Capătul inițial al epicicloidei este poziționat în punctul (x1, y1). Epicicloida este trasată în sensul acelor de ceasornic, iar dacă este bifată căsuța **"direction <"**, atunci epicicloida va fi trasată în sens trigonometric.

OBIECTIVUL 10. ELABORAREA UNUI MODEL DE PREDICȚIE ŞI COMPENSARE A ERORII DE GENERARE A SUPRAFETEI ÎN CAZUL APROXIMARII PRIN POLI A SUPRAFETELOR (CAZUL PROFILARII SUPRAFETELOR ELICOIDALE CILINDRICE DE PAS CONSTANT)

10.1. Algoritmul de modelare pentru aproximarea prin poli a suprafețelor elicoidale cilindrice

Problematica reprofilării sculei generatoare, scula mărginită de o suprafață periferică primară de revoluție (scula disc sau scula cilindro-frontală), pentru compensarea erorii suprafeței generate, poate fi examinată şi în condițiile în care suprafața generată este definită utilizând forma de reprezentare prin poli.

Astfel, pentru o suprafață elicoidală cilindrică şi de pas constant, generată cu o sculă de tip disc, se defineşte generatoarea efectiv măsurată a acesteia, generatoare care diferă de generatoarea țintă, pentru care a fost profilată scula, figura 10. 1, numită generatoare teoretică.

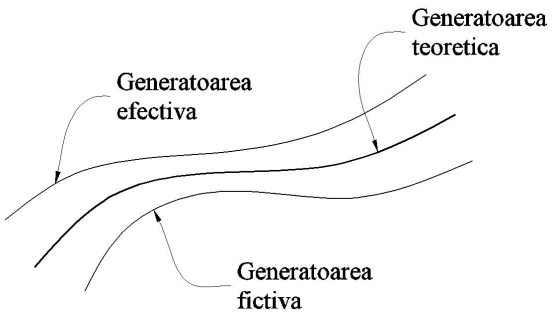


Figura 10. 1. Generatoare ale suprafeței generate: generatoarea țintă, generatoarea efectivă – generatoarea corectată

Modelul analitic al generatoarei teoretice (generatoarea țintă) este definit sub forma unui polinom Bézier, fie acesta de forma:

$$\begin{aligned} X &= A_X \cdot \lambda^2 + \lambda \cdot (1-\lambda) \cdot B_X + (1-\lambda)^2 \cdot C_X; \\ Y &= A_Y \cdot \lambda^2 + \lambda \cdot (1-\lambda) \cdot B_Y + (1-\lambda)^2 \cdot C_Y, \end{aligned} \tag{10.1}$$

cu λ definit între [0,1] şi coeficienții A_X, B_X, ... B_Y definiți conform cu cele trei puncte cunoscute pe generatoare.

Tabelul 10. 1. Coeficienții polinomului de substituire

Coordonate	λ	Coeficienții polinoamelor de substituire	
$A[X_A,Y_A]$	1	$A_X= X_A$	$A_Y= Y_A$
$B[X_B,Y_B]$	1/2	$B_X= 4\cdot X_B-X_A-X_C$	$B_Y= 4\cdot B_Y-Y_A-Y_C$
$C[X_C,Y_C]$	0	$C_X= X_C$	$C_Y= Y_C$

Este evident, aşa cum am arătat în obiectivele anterioare, polinomul de substituire poate avea un grad superior (3, 4 sau chiar mai mare). În acest caz, flexibilitatea şi uşurință de aplicare a metodei se reduce.

Pentru o multitudine de generatoare simple, un polinom de grad 2 sau 3 este satisfăcător, aşa cum s-a demonstrat anterior.

Generatoarea efectivă, rezultată prin măsurarea directă pe suprafața generată, cel mai adesea ca generatoare plană, este cunoscută printr-un număr limitat de puncte, în forma

$$G = \begin{Vmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ \dots & \dots \\ X_i & Y_i \\ \dots & \dots \\ X_n & Y_n \end{Vmatrix}. \tag{10.2}$$

Evident, punctele generatoarei efective, din diferite motive, nu aparțin modelului polinomial.

Ca urmare, procesul de generare urmărind generatoarea țintă este afectat de erori, a căror cauze directe nu pot fi, uşor, decelate.

În acest caz, eventuala corecție la reluarea procesului pe acelaşi semifabricat, dacă mai este posibil, sau pentru generarea aceleiaşi suprafețe, pe alte semifabricate (cazul unei producții de serie), presupune o corecție de sculă prin reprofilarea acesteia pornind de la noua generatoare țintă, diferită de cea teoretică, generatoare pe care o vom denumi *generatoare fictivă*.

Generatoarea fictivă (G_F) se defineşte ca "oglindita" generatoarei efective - G_{E_e} - punct cu punct, în raport cu generatoarea teoretică.

Adică, pentru un punct aparținând generatoarei efective, fie M_i [X_i, Y_i] acesta, îi corespunde ca punct aflat pe generatoarea teoretică, M_j ,punctul aflat la cea mai mică distanță de M_i,

$$\delta_{i,j} = \left\{ \sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2} \right\}_{\min} \tag{10.3}$$

Se defineşte punctul oglindit, al punctului M_i, în raport cu generatoarea teoretică, punctul M_{ij_F} , care este definit de:

$$\begin{vmatrix} X_{M_{ijF}} = X_i + (1+k) \cdot \delta_{i,j} \cdot \cos\beta_{ij}; \\ Y_{M_{ijF}} = Y_i + (1+k) \cdot \delta_{i,j} \cdot \sin\beta_{ij}; \end{vmatrix} \tag{10.4}$$

k este un termen de multiplicare, în mod curent, $0 < k \le 1$;

$$tg\beta_{ij} = \frac{\left|Y_j - Y_i\right|}{\left|X_j - X_i\right|}. \tag{10.5}$$

Totalitatea punctelor M_{ij_F} , astfel definite, determină noua generatoare țintă - generatoarea fictivă, în forma:

$$G_F = \begin{Vmatrix} X_{1,1_F} & Y_{1,1_F} \\ X_{2,2_F} & Y_{2,2_F} \\ \dots & \dots \\ X_{i,j_F} & Y_{i,j_F} \end{Vmatrix}_{(i=1\dots n)(j=1\dots m)} \tag{10.6}$$

Generatoarea fictivă serveşte ca bază pentru modelarea analitică sau în formă discretă, punct cu punct, a suprafeței elicoidale de generat, suprafață ce va servi pentru reprofilarea sculei (scula-disc, scula cilindro-frontală) pentru generarea suprafeței elicoidale.

La reluarea generării, factorii care au generat eroarea, eroarea generatoarei efective față de generatoarea teoretică, acționând la fel, conduc la o nouă generatoare efectivă, mai apropiată de generatoarea teoretică – ținta inițială a generării.

Aproximarea generatoarei fictive

Se propune, în acord cu problematica generală a utilizării polinoamelor Bézier în domeniul profilării sculelor reciproc înfăşurătoare suprafeţelor elicoidale, simplificarea modului de determinare a generatoarei fictive, prin considerarea numai a nodurilor specifice polinoamelor Bézier, care definesc această generatoare, vezi şi figura 10. 2.

Se consideră pe generatoarea efectivă punctele de capăt, fie M_1 $[X_1,Y_1]$ şi M_n $[X_n, Y_n]$ acestea, precum şi două puncte intermediare, M_{i-1} $[X_{i-1},Y_{i-1}]$ şi M_{i+1} $[X_{i+1},Y_{i+1}]$, în jurul punctului reprezentând mijlocul arcului $\widehat{M_1M_n}$.

Acestor puncte le corespund, în conformitate cu algoritmul anterior prezentat, punctele oglindite de pe generatoarea fictivă:

$M_{1,A}$ $[X_{1,A}; Y_{1,A}]$;

$M_{n,C}$ $[X_{n,C}; Y_{n,C}]$;

$M_{i-1,j-1}$ $[X_{i-1,j-1}; Y_{i-1,j-1}]$;

$M_{i+1,j+1}$ $[X_{i+1,j+1}; Y_{i+1,j+1}]$.

Ultimele două puncte, apropiate zonei centrale, vor servi pentru aproximarea nodului mijlociu al polinomului de aproximare al generatoarei fictive.

Pentru polinoamele Bézier de gradul II,

$$\begin{aligned} X &= A_X \cdot \lambda^2 + 2\lambda \cdot (1-\lambda) \cdot B_X + (1-\lambda)^2 \cdot C_X; \\ Y &= A_Y \cdot \lambda^2 + 2\lambda \cdot (1-\lambda) \cdot B_Y + (1-\lambda)^2 \cdot C_Y, \end{aligned} \tag{10.7}$$

care descriu forma generatoarei fictive – G_F, căreia îi aparţin punctele (10.6), se pune problema determinării coeficienţilor A_X, B_X, C_X, A_Y, B_Y, C_Y.

Din sistemul de ecuaţii, determinat din (10.7), pentru considerentele:

$$\lambda = 0, \quad \begin{aligned} X'_A &= C_X; \\ Y'_A &= C_Y; \end{aligned} \tag{10.8}$$

$$\lambda = 1, \quad \begin{aligned} X'_C &= A_X; \\ Y'_C &= A_Y; \end{aligned} \tag{10.9}$$

$$\lambda_{i-1}, \quad \begin{aligned} X_{B'} &= A_X \cdot \lambda_{i-1}^2 + 2\lambda_{i-1} \cdot (1-\lambda_{i-1}) \cdot B'_X + (1-\lambda_{i-1})^2 \cdot C_X; \\ Y_{B'} &= A_Y \cdot \lambda_{i-1}^2 + 2\lambda_{i-1} \cdot (1-\lambda_{i-1}) \cdot B'_Y + (1-\lambda_{i-1})^2 \cdot C_Y; \end{aligned} \tag{10.10}$$

$$\lambda_{i+1}, \quad \begin{aligned} X_{B''} &= A_X \cdot \lambda_{i+1}^2 + 2\lambda_{i+1} \cdot (1-\lambda_{i+1}) \cdot B''_X + (1-\lambda_{i+1})^2 \cdot C_X; \\ Y_{B''} &= A_Y \cdot \lambda_{i+1}^2 + 2\lambda_{i+1} \cdot (1-\lambda_{i+1}) \cdot B''_Y + (1-\lambda_{i+1})^2 \cdot C_Y, \end{aligned} \tag{10.11}$$

se definesc

$$\lambda_{i-1} = \frac{\overline{AB'}}{\overline{AB} + \overline{B'B''} + \overline{B''C}} \text{ şi } \lambda_{i+1} = \frac{\overline{AB} + \overline{B'B''}}{\overline{AB} + \overline{B'B''} + \overline{B''C}} \tag{10.12}$$

precum şi coeficienţii B''_X, B'_X; B''_Y, B'_Y.

Astfel, se poate aproxima nodul B determinând constantele

$$B_X = \frac{B'_X + B''_X}{2}, \; B_Y = \frac{B'_Y + B''_Y}{2}. \tag{10.13}$$

Se identifică, în acest fel, un polinom Bézier, care aproximează generatoarea fictivă - G_{Fs}, ca substituent al generatoarei fictive, trasată punct cu punct.

Problema poate fi concepută şi într-un alt mod. Se substituie generatoarea fictivă $G_{A'B'C'}$ cu un polinom Bézier de grad inferior (gradul II sau III), astfel că generatoarea fictivă poate fi privită ca oglindita acestui polinom în raport cu generatoarea teoretică.

Astfel, s-ar pune problema determinării erorii de aproximare a generatoarei fictive $G_{A''B''C''}$, în acest mod determinată, cu generatoarea fictivă oglindită punct cu punct (vezi figura 10. 2).

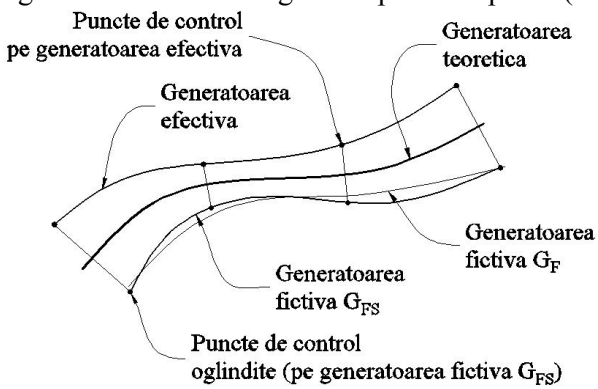


Figura 10. 2. Generatoare fictivă şi efectivă substituite cu polinoame

Este evident, problema poate fi tratată în mod similar şi pentru polinoame de substituire de gradul III, pentru o mai bună apreciere a punctelor de pe profiluri.

10.2. Elaborarea de produse soft specifice

Compensarea erorii generării melcului de compresor elicoidal

Modelului discret al profilului teoretic al generatoarei compresorului elicoidal, vezi tabelul 10.2, îi corespunde un profil măsurat (efectiv), vezi tabelul 10.3.

Tabelul 10. 2. Coordonate ale profilului teoretic — arcul $\overset{\frown}{AHG}$ (în corelaţie cu figura 9.16)

X[mm]	Y[mm]
31.27239	4.65301
31.38198	5.36422
31.5133	6.07175
31.66712	6.77472
31.8445	7.47212
32.04638	8.16282
32.27317	8.84574
32.52581	9.51952
32.8041	10.1831
33.10882	10.835
33.43917	11.4743
33.79571	12.0993
34.17711	12.7095
34.58361	13.3033
35.0135	13.8804
35.46662	14.4394
35.94113	14.9803
36.43628	15.5025
36.95042	16.006
37.48202	16.491
38.02985	16.9576
38.59189	17.407
39.16501	17.8422
39.71991	18.3002

40.24701	18.79
40.74927	19.3053

Tabelul 10. 3. Model al profilului efectiv (măsurat) al rotorului — arcul $\overset{\frown}{AHG}$

X[mm]	Y[mm]
31.272	4.653
31.381	5.366
31.528	6.168
31.700	6.956
31.916	7.748
32.144	8.528
32.413	9.281
32.730	10.050
33.057	10.790
33.428	11.497
33.806	12.177
34.240	12.899
34.732	13.522
35.224	14.203
35.748	14.780
36.250	15.399
36.836	15.955
37.413	16.513
38.066	17.031
38.718	17.547
39.330	18.084
39.981	18.593
40.516	19.181
40.702	19.322

Pentru profilul melcilor de compresor elicoidal, vezi figura 10. 3, se cunosc coordonatele aparținând profilului teoretic, vezi tabelul 10. 2 şi corelat cu figura 9.16

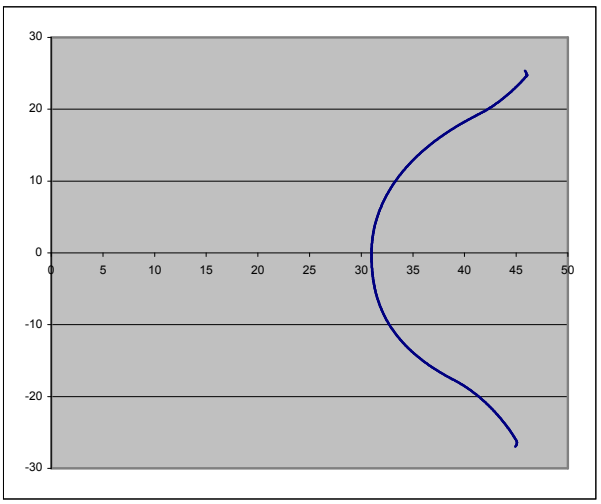


Figura 10. 3. Profilul melcului condus ($\overset{\frown}{GHA}$—ansamblul de profiluri Bezier, în corelație cu figura 9.16)

Se realizează compensarea erorii de profilare prin oglindirea profilului măsurat, față de profilul teoretic. Oglindirea, determinarea generatoarei fictive, se va face în două moduri:

A *Profil oglindit* (generatoare fictivă) – *obținută prin simetrizarea tuturor punctelor față de profilul teoretic* şi aproximarea ulterioară a punctelor măsurate printr-un polinom Bezier de grad III;

B *Profil oglindit – obținut prin interpolarea punctelor măsurate* şi oglindirea doar a nodurilor polinomului, 3 sau 4 puncte, rezultând astfel un alt polinom Bezier, mult mai simplu de determinat.

Tabelul 10. 4. Comparație între profilurile A şi B

Lambda	Profil oglindit – A		Profil oglindit B		Eroarea[mm]
	X[mm]	Y[mm]	X[mm]	Y[mm]	
0.0000	31.2732	4.6528	31.2732	4.6528	0.0000
	31.3768	5.5378	31.3792	5.5427	0.0054
	31.5318	6.4153	31.5361	6.4171	0.0047
	31.7371	7.2824	31.7416	7.2758	0.0080
	31.9909	8.1366	31.9990	8.1353	0.0083
	32.2911	8.9756	32.3024	8.9775	0.0114
	32.6353	9.7975	32.6419	9.7857	0.0134
0.3330	32.8853	10.3299	32.8930	10.3203	0.0123
	33.0209	10.6008	33.0291	10.5925	0.0117
	33.4450	11.3845	33.4552	11.3808	0.0109
	33.9049	12.1478	33.9179	12.1502	0.0131
	34.3979	12.8902	34.4044	12.8854	0.0080
	34.9213	13.6114	34.9214	13.6017	0.0097
	35.4726	14.3115	35.4782	14.3127	0.0057
	36.0497	14.9906	36.0502	14.9895	0.0012
0.6660	36.2393	15.2036	36.2423	15.2062	0.0040
	36.6503	15.6489	36.6461	15.6462	0.0050
	37.2726	16.2868	37.2764	16.2952	0.0092
	37.9149	16.9046	37.9137	16.9100	0.0055
	38.5755	17.5027	38.5681	17.5036	0.0075
	39.2531	18.0816	39.2512	18.0869	0.0056
	39.9464	18.6415	39.9474	18.6472	0.0058
1.0000	40.6542	19.1829	40.6544	19.1841	0.0012

MAX ERROR:0.0138

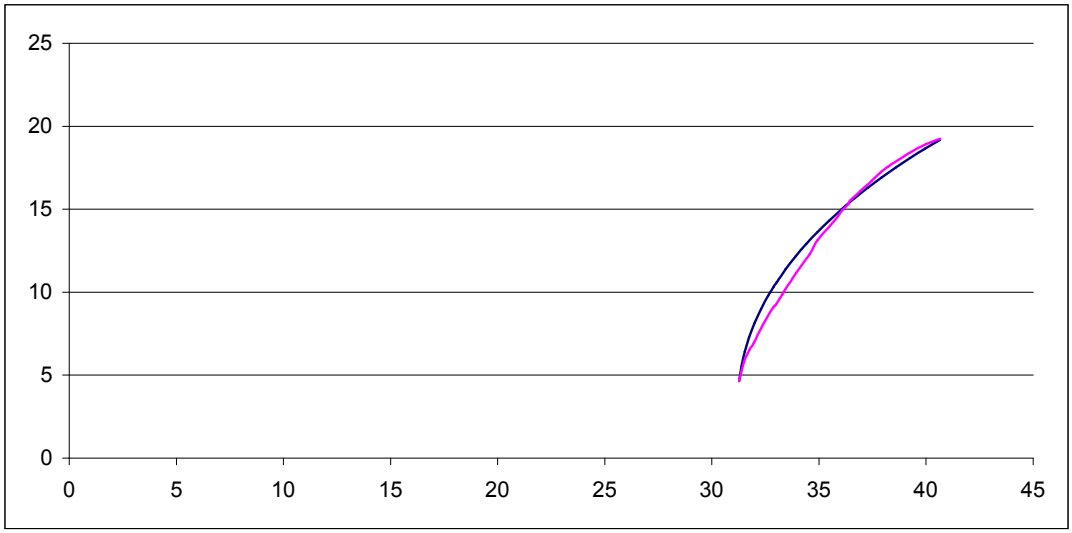


Figura 10. 4. Albastru – Profilul oglindit A;
Rosu - Profil oglindit – B (eroarea x50)

Tabelul 10. 5. Comparație între profilurile sculelor disc profilate pe baza profilurilor A şi B

λ	Profilul axial al sculei disc pentru generatoare fictivă în condițiile A		Profilul axial al sculei disc pentru generatoare fictivă în condițiile B		Eroare [mm]
	X [mm]	Y [mm]	X [mm]	Y [mm]	
0.0000	68.4706	3.0096	68.4716	3.0062	0.0035
	68.1475	3.6824	68.1534	3.6842	0.0061
	67.7671	4.3245	67.7762	4.3279	0.0097
	67.3318	4.9308	67.3408	4.9369	0.0109
	66.8464	5.4976	66.8584	5.5003	0.0123
	66.3164	6.0232	66.3221	6.0306	0.0093
	65.7482	6.5071	65.7570	6.5096	0.0092
0.3330	65.3553	6.8044	65.3648	6.8049	0.0096
	65.1478	6.9506	65.1449	6.9593	0.0091
	64.5210	7.3560	64.5267	7.3569	0.0058
	63.8727	7.7260	63.8696	7.7303	0.0053
	63.2071	8.0640	63.2037	8.0671	0.0046
	62.5277	8.3735	62.5339	8.3715	0.0065
	61.8375	8.6578	61.8363	8.6586	0.0014
	61.1386	8.9202	61.1429	8.9190	0.0045
0.6660	60.9134	9.0000	60.9102	9.0016	0.0036
	60.4329	9.1639	60.4285	9.1659	0.0048
	59.7220	9.3920	59.7261	9.3916	0.0040
	59.0073	9.6076	59.0100	9.6080	0.0027
	58.2897	9.8136	58.2842	9.8167	0.0063
	57.5703	10.0133	57.5676	10.0157	0.0037
	56.8501	10.2099	56.8501	10.2115	0.0016
1.0000	56.1300	10.4069	56.1371	10.4064	0.0071

LAMBDA MAX ERROR:0.208
MAX ERROR: 0.0127

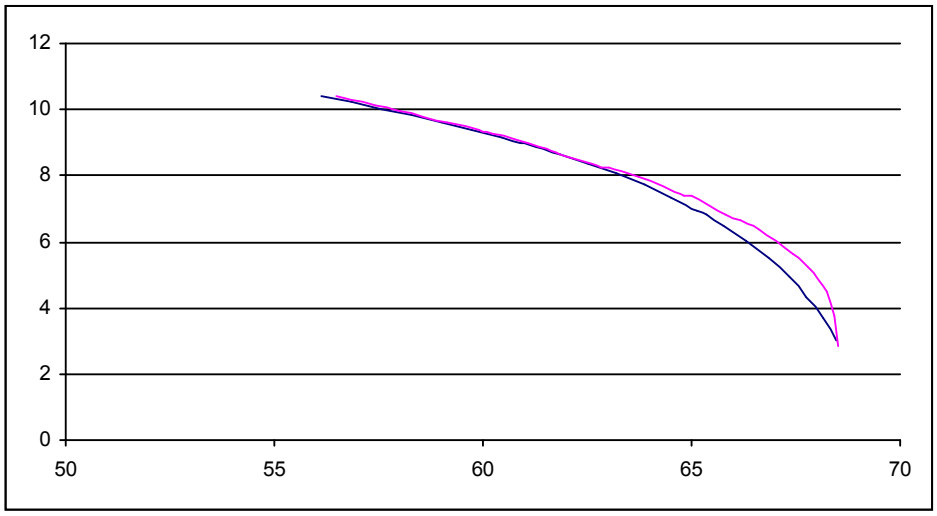


Figura 10. 5. Albastru – Scula disc - Profilul oglindit A (eroarea x50);

În figura 10. 5, sunt reprezentate cele două forme ale generatoarei fictive în condițiile oglindirii tuturor punctelor generatoarei efective (forma A) sau numai a punctelor de control ale polinomului Bezier substitutiv (forma B), constituind forma simplificată a generatoarei, mult mai simplu de determinat.

În baza algoritmilor cunoscuți, se determină profilurile axiale ale sculelor disc, generatoare a suprafețelor elicoidale fictive, modelate în baza celor două generatoare fictive, anterior determinate.

În tabelul tabelul 10. 5, sunt prezentate forma secțiunilor axiale ale sculelor disc pentru cele două situații şi a diferenței între ele.

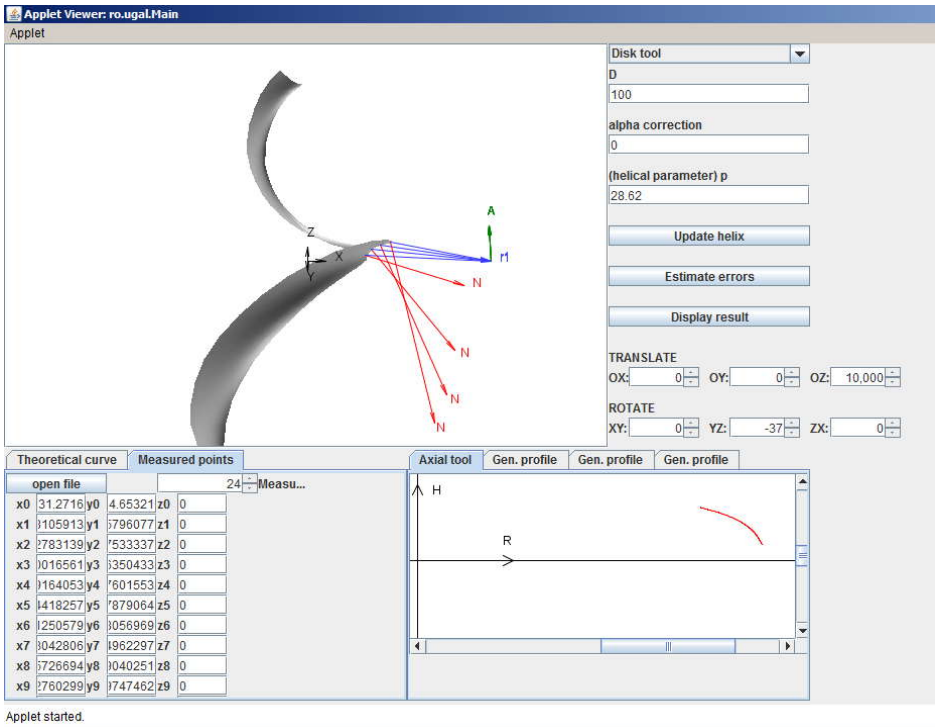


Figura 10. 6. Rosu – Scula disc - Profil oglindit – A (eroarea x50)

În figura 10. 6, este prezentat un applet al programului dedicat problemei anterior prezentată, în limbajul de programare Java, realizat în cadrul programului de cercetare.

Concluzii

Modelul de compensare a erorii de generare prin introducerea noțiunii de generatoare fictivă (generatoare care se obține din generatoarea efectiv măsurată) permite o reprofilare a sculei, pentru exemplul nostru, cazul sculei disc, care, la reluarea prelucrării, poate genera o suprafață (o generatoare a acesteia) mai apropiată de suprafața țintă inițială.

Generatoarea fictivă poate fi substituită pentru un număr redus de puncte ale acesteia (3 sau 4 puncte) printr-un polinom Bezier de grad inferior (gradul 2 sau 3) — metodă simplificată.

S-a dovedit că, generatoarea fictivă obținută în baza oglindirii tuturor punctelor generatoarei efective şi generatoarea fictivă obținută prin metoda simplificată, sunt foarte apropiate ca formă, conducând la profiluri ale sculelor disc, generatoare ale țintelor fictive, tehnic identice.

OBIECTIVUL 11. SINTEZA UNOR PRODUSE SOFT SPECIALIZATE, PENTRU PROFILAREA SCULELOR GENERATOARE A SUPRAFEȚELOR ELICOIDALE CILINDRICE COMPLEXE, BAZATE PE REPREZENTAREA ÎN FORMA DISCRETA A SUPRAFEȚELOR (REPREZENTARE POLIEDRALĂ SAU PRIN POLI)

Problematica profilării sculelor care generează prin înfăşurare suprafețe elicoidale este bine cunoscută, soluția problemei făcând apel la teoremele fundamentale ale înfăşurării suprafețelor, pentru cazul în care acestea sunt reprezentate în forme analitice, teorema 1 Olivier.

Adesea, apare problema unor reprezentări neanalitice a suprafețelor, în legătură cu aplicațiile de inginerie inversă, în care suprafețele efective ale semifabricatelor sunt cunoscute prin măsurare directă pe maşini de măsurat 3 D.

Se pune, în acest fel, problema aproximării suprafeței, astfel cunoscute, şi înlocuirea acesteia cu o suprafață (ansamblu de suprafețe) analitică, care să reprezinte cea mai bună aproximație a ansamblului de puncte măsurate, care să permită, pe această cale, utilizarea metodelor analitice cunoscute, în vederea profilării sculelor mărginite de suprafețe periferice primare de revoluție, reciproc înfăşurătoare suprafeței elicoidale cunoscute în formă discretă.

Multiple soluții sunt cunoscute în literatură pentru o astfel de apreciere a formei suprafeței, identificarea acesteia şi aproximarea ulterioară, presupunând cunoscut tipul de suprafață, cu aplicații specifice în proiectarea ulterioară a sculelor generatoare a acestora.

Sunt prezentate soluții specifice construcției suprafeței periferice primare a sculei generatoare.

În lucrare, se propune o metodă de aproximare a unei suprafețe efectiv măsurate printr-un ansamblu de suprafețe plane, şi a unui produs soft specific, realizat în limbaj Java, în scopul profilării sculelor disc reciproc înfăşurătoare cu suprafața efectivă, substituită prin acest ansamblu de suprafețe – metoda poliedrală.

11.1. Metoda prezentării poliedrale a suprafețelor

Suprafețele (elicoidală, cilindrică sau de revoluție), aşa cum rezultă în urma măsurării prin exploatare cu un sistem de palpare, care determină coordonatele succesive ale punctelor acestora, figura 11. 1, pot fi privite ca fiind formate dintr-o rețea de puncte distincte în lungul liniilor de măsurare.

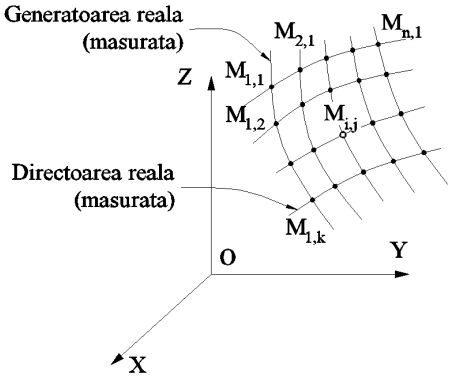


Figura 11. 1. Suprafața reală (determinată prin măsurarea punctelor)

Facem observația că distribuția punctelor măsurate în lungul generatoarelor efective trebuie a fi suficient de densă pentru a descrie suprafața în limitele unei anumite precizii de măsurare, acceptată ca riguroasă din punct de vedere tehnic.

Deşi rețeaua de puncte formată pe suprafața efectiv măsurată nu este o rețea cu elemente uniforme, algoritmul pentru determinarea normalei la suprafață nu este afectat, dacă numărul de puncte este suficient de mare.

În sensul prezentat anterior, o generatoare efectivă „j" a suprafeței poate fi reprezentată printr-o matrice de forma:

$$G = \begin{Vmatrix} X_{1,j} & X_{2,j} & X_{3,j} & \dots & X_{k,j} \\ Y_{1,j} & Y_{2,j} & Y_{3,j} & \dots & Y_{k,j} \\ Z_{1,j} & Z_{2,j} & Z_{3,j} & \dots & Z_{k,j} \end{Vmatrix}^{T} . \tag{11.1}$$

Ținând seama de (11.1), pentru rețeaua de puncte reprezentând o zonă a suprafeței se acceptă exprimarea:

$$\Sigma_{efectiv} = \left\{ \begin{Vmatrix} X_{1,j} & X_{2,j} & \dots & X_{k,j} \\ Y_{1,j} & Y_{2,j} & \dots & Y_{k,j} \\ Z_{1,j} & Z_{2,j} & \dots & Z_{k,j} \end{Vmatrix}^{T} \right\}_{i} ; \quad i = 1,2,\dots, j,\dots m. \tag{11.2}$$

Normala intr-un punct oarecare al suprafeței efective (11.2), fie $M_{i,j}$ acesta, se defineşte ca fiind normala la una dintre fețele „poliedrului" determinat de punctele: $M_{i,j}$; $M_{i,j-1}$; $M_{i+1,j}$ etc, figura 11. 2.

Este evident că, în punctul considerat, $M_{i,j}$, se pot defini patru normale, câte una la fiecare din cele patru fețe ale poliedrului având ca vârf punctul considerat.

De exemplu, pornind de la coordonatele vecine ale punctului $M_{i,j}$, spre exemplu:

$$M_{i,j-1} = \begin{Vmatrix} X_{i,(j-1)} \\ Y_{i,(j-1)} \\ Z_{i,(j-1)} \end{Vmatrix}; \tag{11.3}$$

$$M_{i,j} = \begin{Vmatrix} X_{i,j} \\ Y_{i,j} \\ Z_{i,j} \end{Vmatrix}; \tag{11.4}$$

$$M_{(i+1),j} = \begin{Vmatrix} X_{(i+1),j} \\ Y_{(i+1),j} \\ Z_{(i+1),j} \end{Vmatrix}, \tag{11.5}$$

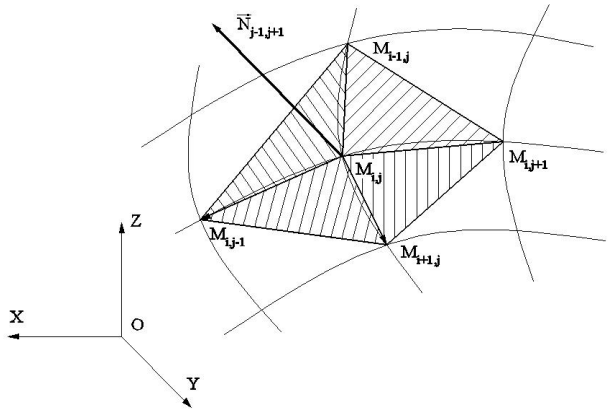


Figura 11. 2. Normala la suprafaţa poliedrală

se pot defini vectorii:

$$\overrightarrow{M_{i,j}M}_{i,j-1} = (X_{i,j} - X_{i,j-1})\cdot\vec{i} + (Y_{i,j} - Y_{i,j-1})\cdot\vec{j} + (Z_{i,j} - Z_{i,j-1})\cdot\vec{k} \tag{11.6}$$

şi

$$\overrightarrow{M_{i,j}M}_{(i+1),j} = (X_{i,j} - X_{i+1,j})\cdot\vec{i} + (Y_{i,j} - Y_{i+1,j})\cdot\vec{j} + (Z_{i,j} - Z_{i+1,j})\cdot\vec{k}\,. \tag{11.7}$$

Astfel, normala la suprafaţa plană determinată de aceste puncte este

$$\vec{E}_{i,j}^{(i+1),(j-1)} = \overrightarrow{M_{i,(j-1)}M}_{i,j} \times \overrightarrow{M_{i,j}M}_{(i+1),j} \tag{11.8}$$

sau, sub formă de determinant,

$$\vec{E}_{i,j}^{(i+1),(j-1)} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ (X_{i,(j-1)} - X_{i,j}) & (Y_{i,(j-1)} - Y_{i,j}) & (Z_{i,(j-1)} - Z_{i,j}) \\ (X_{(i+1),j} - X_{i,j}) & (Y_{(i+1),j} - Y_{i,j}) & (Z_{(i+1),j} - Z_{i,j}) \end{vmatrix}. \tag{11.9}$$

În mod similar, se definesc normalele şi la celelalte suprafeţe ale poliedrului cu vârful în $M_{i,j}$.

Un algoritm de parcurgere a generatoarelor succesive ale zonei efectiv măsurate, vezi (11.2), va permite determinarea vectorului normal la suprafeţele plane poliedrelor, astfel formate pe suprafaţa efectiv măsurată (11.2).

Modalitatea de reprezentare a suprafeţei efectiv măsurate poate permite determinarea curbei caracteristice a suprafeţei, în mişcarea absolută a acesteia, rotaţie sau translaţie, în legătură cu tipul de sculă generatoare căutată.

11.2. Ajustarea formei suprafeţei măsurate

Există posibilitatea, atunci când numărul de puncte măsurate pe suprafaţă nu poate fi foarte mare şi când există suspiciunea că forma poliedrică de substituire a suprafeţei efectiv măsurate se îndepărtează mult de la forma reală a suprafeţei, conducând la o variaţie neuniformă a parametrilor directori ai normalei la suprafeţele poliedrale de substituire, să facă o ajustare a formei suprafeţei măsurate (fitting), încât să nu apară discontinuităţi în descrierea acesteia.

Aceasta se poate realiza în baza unui produs soft specializat, pornind de la norul de puncte măsurate pe suprafaţa efectivă, şi obţinerea unui nou nor de puncte, aflat de această dată pe suprafaţa ajustată, nor de puncte care nu în integralitatea punctelor sale reprezintă puncte efectiv măsurate pe suprafaţă.

Odată acest nou număr de puncte obţinut prin ajustare, matricea de tipul (11.1) se modifică, definindu-se un nou grid de aproximare a suprafeţei şi noi aproximări ale formelor vectorilor

pornind din punctul $M_{i,j}$, vezi (11.5) şi (11.6), şi figura 11. 3, sau, similar, din punctul $m_{i,j}$, pentru gridul ajustat.

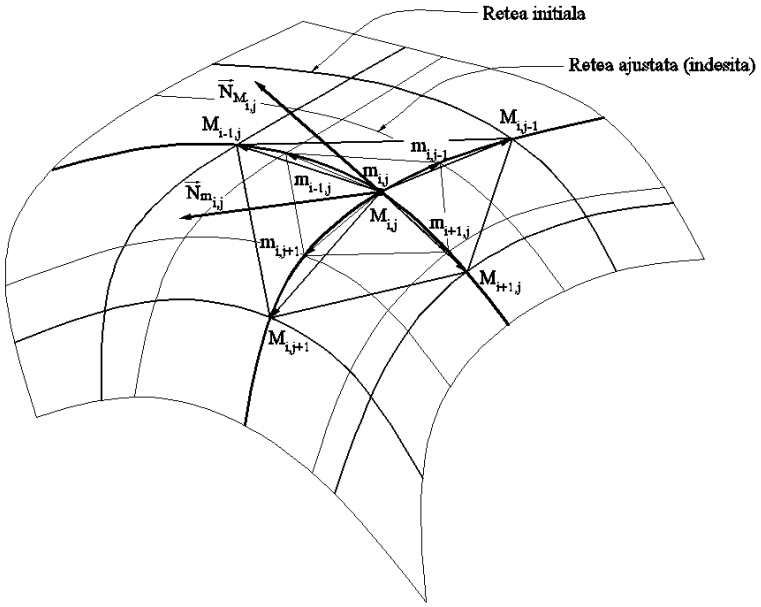


Figura 11. 3. Suprafeţe poliedrale: grid iniţial ($M_{i,j}$; $M_{i+1,j}$; ...); grid ajustat ($m_{i,j}$; $m_{i+1,j}$ ...)

În figura 11. 3, cu $M_{i,j}$, $M_{i,j-1}$, $M_{i,j+1}$ ... s-au notat nodurile gridului iniţial (măsurat) al suprafeţei efective şi cu $m_{i,j}$, $m_{i,j-1}$, $m_{i,j+1}$ nodurile gridului ajustat.

De asemenea, s-au notat cu $\vec{N}_{M_{i,j}}$ şi $\vec{N}_{m_{i,j}}$ vectorii normalelor la suprafeţele poliedrale cu vârful în punctul curent $M_{i,j}$ ($m_{i,j}$), pentru cele două forme de grid considerate, iniţial (măsurat) şi ajustat.

11.3. Profilarea sculei disc

11.3.1 Curba caracteristică a suprafeţei exprimată în formă discretă

Se urmăreşte a se determina forma curbei caracteristice, în mişcarea de rotaţie a suprafeţei de generat exprimată în formă discretă, (vezi şi figura 11. 3), în jurul unei axe, fixă şi definită ca poziţie, reprezentând axa viitoarei scule mărginită de o suprafaţă periferică primară de revoluţie – scula disc sau scula cilindro – frontală.

În figura 11. 4, sunt prezentate sistemele de referinţă şi poziţia axei viitoarei scule – disc.

Conform teoremei Novicov, condiţia ca punctul $M_{i,j}$, de pe suprafaţa Σ, să aparţină curbei caracteristice este determinată de intersecţia normalei la Σ, în acest punct, cu axa sculei – disc.

Se definesc sistemele de referinţă:

XYZ este sistemul în care este definită suprafaţa măsurată;

$X_1Y_1Z_1$ – sistemul de referinţă solidar axei sculei disc.

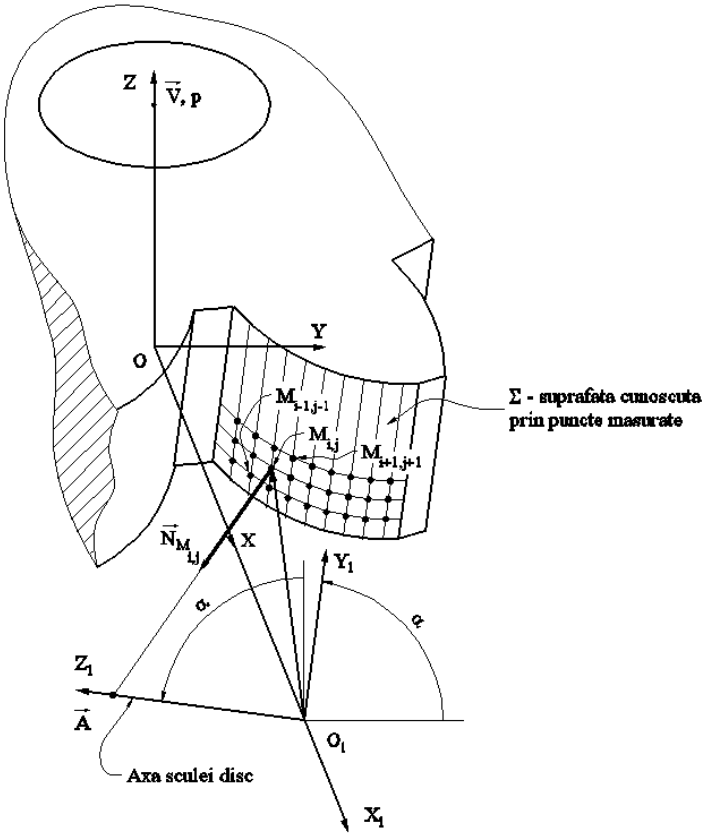


Figura 11. 4. Scula disc - sisteme de referinţă

a – distanţa între axa suprafeţei Σ (axa $\vec{V}$) şi axa sculei disc.

Mărimile α şi a sunt mărimi tehnologice. Parametrul unghiular α, pentru un canal elicoidal aparţinând unei suprafeţe elicoidale cilindrice şi de pas constant, suprafaţă cunoscută în formă discretă, se determină din condiţia ca axa $\vec{A}$ să fie perpendiculară pe una dintre elicele caracteristice ale suprafeţei, în mod obişnuit, pe elicea corespunzătoare diametrului maxim al suprafeţei.

Mărimea a, distanţa între cele două axe, $\vec{A}$ şi $\vec{V}$, se defineşte din considerente constructiv tehnologice: dimensiunea transversală a suprafeţei de generat şi diametrul exterior al viitoarei scule disc.

Condiţia ca punctul curent $M_{i,j}$, al reţelei efective pe suprafaţa Σ, să aparţină curbei de contact cu suprafaţa de revoluţie de axă $\vec{A}$ este ca normala la una dintre feţele poliedrului, cu vârful în punctul $M_{i,j}$, să intersecteze axa $\vec{A}$.

Altfel spus, dacă se defineşte vectorul de poziţie $\vec{r}_1$, vezi figura 11. 4, ca vector care uneşte originea O_1 cu punctul $M_{i,j}$,

$$\vec{r}_1 = (X_{i,j} - a)\cdot\vec{i} + Y_{i,j}\cdot\vec{j} + Z_{i,j}\cdot\vec{k}, \tag{11.10}$$

în care $X_{i,j}$, $Y_{i,j}$, $Z_{i,j}$ sunt date de (11.2), condiţia de intersecţie cu axa $\vec{A}$ a normalei (11.9), poate fi scrisă în forma

$$\left|\left(\vec{A},\vec{r}_1\vec{N}_{\Sigma_{i,j}}\right)\right| \le \varepsilon \tag{11.11}$$

în care, ε este o valoare pozitivă, suficient de mică.

Condiţia (11.11) trebuie testată pentru toate cele patru feţe laterale ale poliedrului cu vârful în punctul $M_{i,j}$. Evident, se acceptă acea normală, pentru care condiţia (11.11) este cea mai apropiată de zero, în valoare absolută.

În acest fel, stabilindu-se ”fața” poliedrului care corespunde, cel mai îndeaproape, condiției de înfăşurare (11.11), se decide ”avansul”, pentru testarea următorului punct în definirea curbei caracteristice.

Ansamblul punctelor $M_{i,j}$ care satisfac condiția (11.11) reprezintă caracteristica suprafeței elicoidale, în formă discretă, şi, implicit, caracteristica suprafeței de revoluție, care constituie suprafața periferică primară a sculei – disc.

În principiu, caracteristica C_S, astfel determinată, poate căpăta o reprezentare de forma

$$C_S = \left\{ \begin{pmatrix} X_{i,j} \\ Y_{i,j} \\ Z_{i,j} \end{pmatrix}^T \right\}, \ (i = 1,...,n, j = 1,...,m). \tag{11.12}$$

Suprafața periferică primară a sculei disc se obține prin rotirea curbei caracteristice (11.12) în jurul axei $\vec{A}$, axa sculei disc.

Se poate defini curba caracteristică (11.12) în sistemul de referință al sculei disc, vezi şi figura 11. 4, prin transformarea

$$X_1 = \alpha \cdot (X - a) \tag{11.13}$$

în care:

$$\alpha = \begin{Vmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{Vmatrix}; \tag{11.14}$$

este matricea transformării ortogonale între versorii axelor sistemului $X_1Y_1Z_1$, față de XYZ;

$$a = \begin{Vmatrix} a \\ 0 \\ 0 \end{Vmatrix}, \tag{11.15}$$

matricea formată cu coordonatele originii O_1, în sistemul XYZ.

Astfel, ținând seama de (11.12), se poate exprima curba caracteristică, în formă discretă, în sistemul $X_1Y_1Z_1$, în forma:

$$C_{1S} = \left\{ \begin{pmatrix} X_{1_{i,j}} \\ Y_{1_{i,j}} \\ Z_{1_{i,j}} \end{pmatrix} \right\}^T, \quad (i = 1...n), (j = 1...m). \tag{11.16}$$

Prin rotirea curbei caracteristice (11.16) în jurul axei $\vec{A}$ (axa Z_1).

$$\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}^T \cdot \begin{pmatrix} X_{1_{i,j}} \\ Y_{1_{i,j}} \\ Z_{1_{i,j}} \end{pmatrix}, \tag{11.17}$$

se obține forma suprafeței periferice primare a sculei disc, exprimată cu o familie de cercuri.

Secțiunea axială a sculei disc se obține din (11.17) în forma:

$$S_A \left\| \begin{matrix} R = \sqrt{X^2_{1_{i,j}} + Y^2_{1_{i,j}}}; \\ H = Z_{1_{i,j}}. \end{matrix} \right. \tag{11.18}$$

11.3.2. Aproximarea punctelor pe suprafața măsurată

Se consideră suprafața măsurată a flancului suprafeței elicoidale cilindrice — flancul evolventic al roții dințate, vezi figura 11. 5.

Pe maşina de măsurat în coordonate 3D MicroHite, au fost determinate, prin măsurare directă, coordonate ale punctelor de pe generatoarele succesive ale flancului, vezi figura 11. 5.

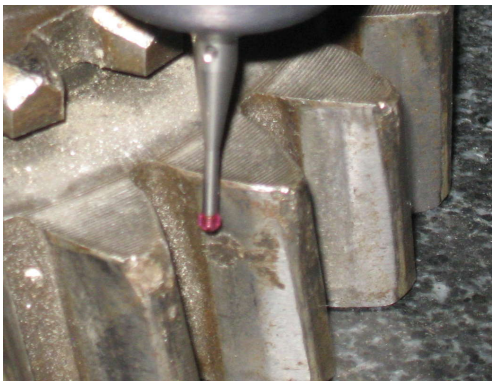


Figura 11. 5. Măsurarea roții dințate cu dinți înclinați

În tabelul 11. 1, sunt prezentate coordonate ale punctelor aparținând generatoarelor succesive măsurate pe suprafață (Z=const.).

Tabelul 11. 1. Puncte discrete măsurate pe generatoare succesive

Linie *i*	**Nr. crt.**	**X [mm]**	**Y [mm]**	**Z [mm]**
1	**1**	-4.53	-65.313	-0.004
	2	-4.62	-65.727	-0.004
	⋮	⋮	⋮	⋮
	19	-8.948	-73	-0.004
	20	-9.307	-73.436	-0.004
2	**1**	-4.127	-65.196	-1.341
	2	-4.17	-65.548	-1.341
	⋮	⋮	⋮	⋮
	19	-8.457	-73.35	-1.341
	20	-8.778	-73.766	-1.341
⋮	⋮	⋮	⋮	⋮
10	**1**	-4.1	-65.37	-1.590
	2	-4.294	-66.134	-1.590
	⋮	⋮	⋮	⋮
	19	-8.558	-73.565	-1.591
	20	-9.063	-74.185	-1.591

Ansamblul generatoarelor succesive formează suprafața discretă a flancului de generat, vezi figura 11. 6.

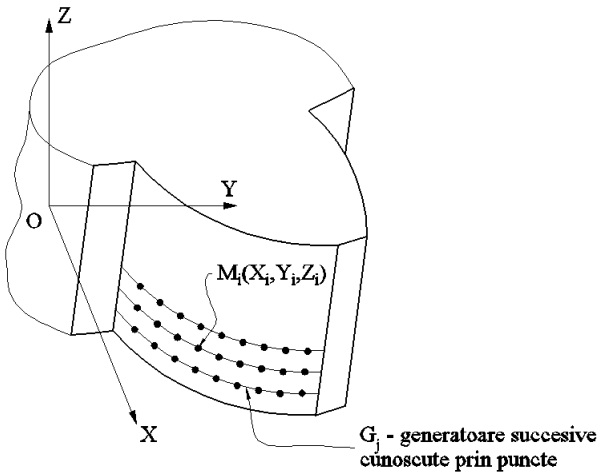


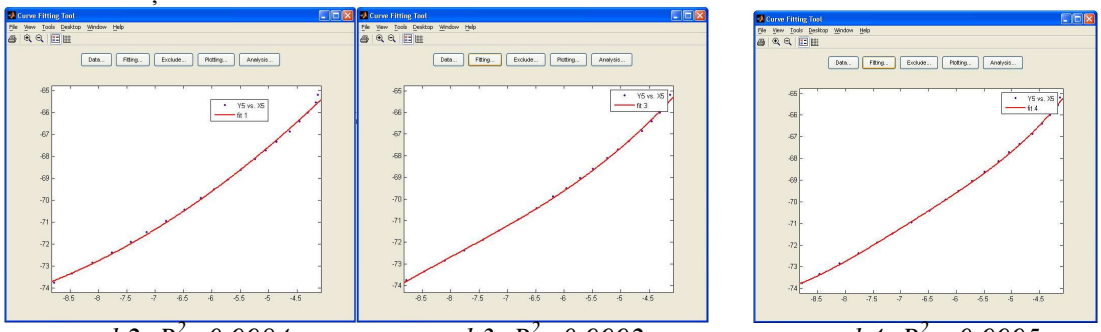
Figura 11. 6. Puncte măsurate pe flancul danturii

Este evident că, suprafața măsurată nu este o suprafață netedă. Este necesară netezirea acesteia, pentru o interpretare riguroasă a datelor măsurate.

Se propune aproximarea fiecărei generatoare printr-o ecuație polinomială, care să îndeplinească următoarele condiții :- indicele R^2 (adjusted R-square) să fie cât mai apropiat de 1;

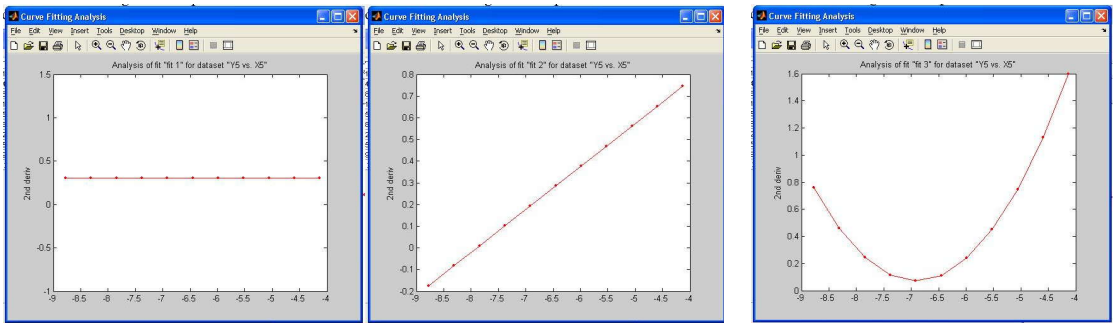
- derivata de ordinul 2 a polinomului de substituție să fie o linie dreaptă pentru a fi evitate punctele de pe generatoare in care există variații semnificative ale curburii.

În figura 11. 7 şi figura 11. 8, sunt prezentate formele polinoamelor de substituire, pentru aproximarea datelor, precum şi derivatele de ordinul unu şi doi ale acestor polinoame, în diverse puncte ale funcției de substituire.



grad 2: R^2=0.9984; *grad 3: R^2=0.9992;* *grad 4: R^2 =0.9995;*

Figura 11. 7. Forma funcției de substituire pentru diferite grade ale polinomului



gradul 2; *gradul 3;* *gradul 4;*

Figura 11. 8. Derivata de ordinul II

Din analiza formelor prezentate, se poate concluziona că, indicele R^2 are cea mai apropiată valoare de 1 pentru un polinom de aproximare de ordinul 2 pentru care, în acelaşi timp, derivata de ordinul doi este liniară, fiind eliminată existența punctelor în care apar variații importante ale curburii.

Notă: Evident, pentru forme diferite ale suprafeței măsurate, polinomul de substituire va avea forme diferite.

Pentru evaluarea polinomului de substituire a fost utilizat programul MatLab.

În acest mod, toate generatoarele suprafeței au fost aproximate prin polinoame de substituire. Facem precizarea că, în funcție de coordonatele măsurate, polinoamele de substituire pot avea grade diferite.

Forma suprafeței substituite poate fi procesată cu programul MatLab, realizându-se o rețea mai deasă de puncte. În figura 11.9, este prezentată o captură de ecran a suprafeței netezite.

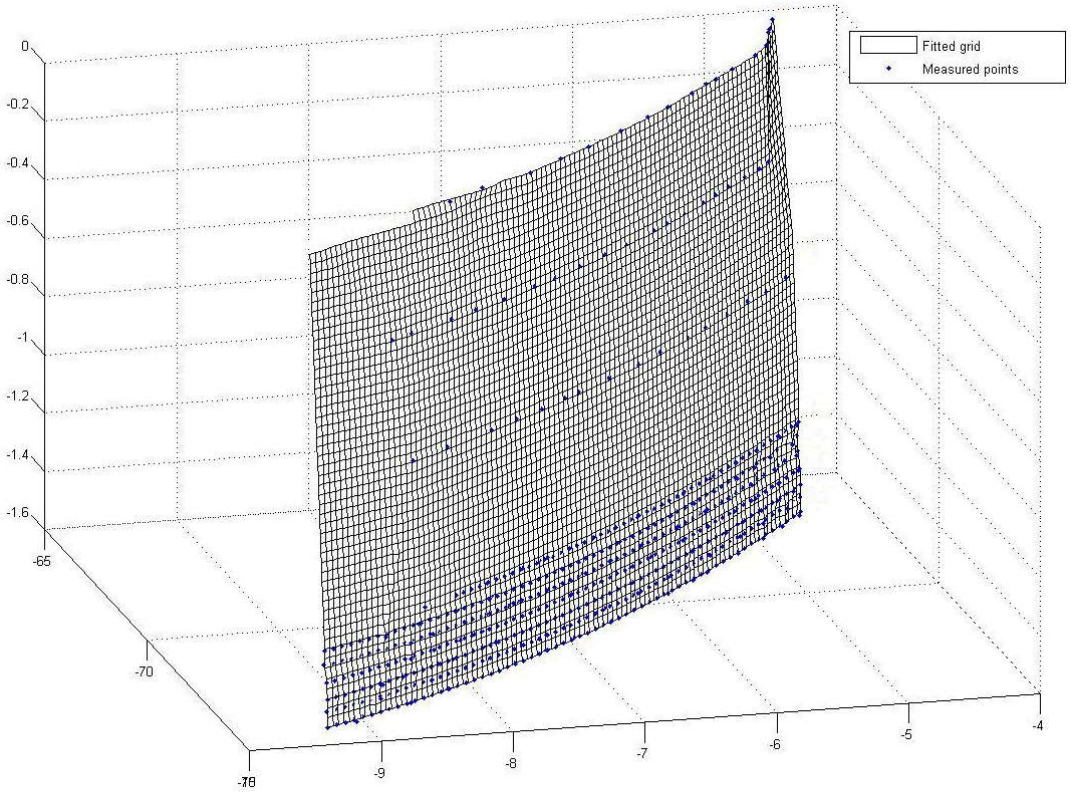


Figura 11. 9. Forma suprafeței de substituire

În tabelul 11. 2, sunt prezentate coordonatele punctelor de pe suprafața care aproximează norul de puncte efectiv măsurat.

Se determină, în concordanță cu algoritmul prezentat, curba caracteristică a suprafeței elicoidale, veyi figura 11.10

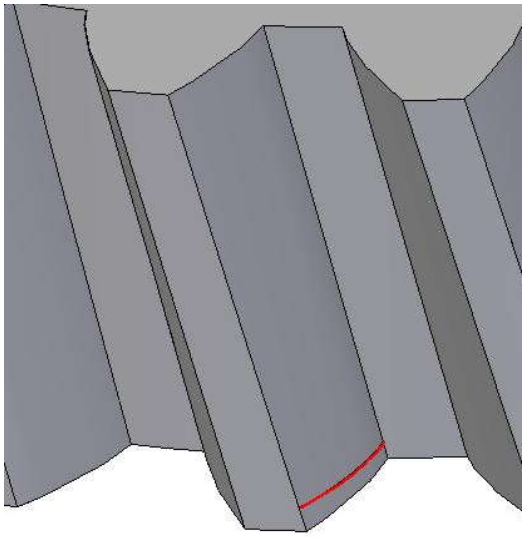


Figura 11. 10. Curba caracteristică la generarea cu scula-disc

Tabelul 11. 2. Coordonatele punctelor generatoarelor aproximate

Linia *i*	Nr. crt.	X [mm]	Y [mm]	Z [mm]
1	**1**	-8.404	-72.476	-0.005
	2	-8.3678	-72.436	-0.005
	⋮	⋮	⋮	⋮
	100	-4.8182	-66.586	-0.005
	101	-4.782	-66.49	-0.005
2	**1**	-8.404	-72.479	-0.00817
	2	-8.3678	-72.439	-0.00817
	100	-4.8182	-66.588	-0.00817
	101	-4.782	-66.492	-0.00817
⋮	⋮	⋮	⋮	⋮
51	**1**	-8.404	-73.359	-1.59
	2	-8.3678	-73.311	-1.59
	⋮	⋮	⋮	⋮
	100	-4.8182	-67.366	-1.59
	101	-4.782	-67.292	-1.59

Datele de intrare pentru programul de calcul realizat în limbajul de programare Java sunt: p=1918.5 mm; D_{ex}=150 mm; z= 26 dinți.

11.3.3. Secțiunea axială a sculei disc

În baza algoritmului prezentat se poate calcula secțiunea axială a sculei disc.

În figura 11. 10, este reprezentată curba caracteristică determinată pe suprafața substitutivă a suprafeței măsurate.

În

tabelul 11. 3, sunt prezentate coordonatele punctelor de pe curba caracteristică pentru o scula disc având diametrul exterior de 60 mm.

Tabelul 11. 3. Coordonate pe curba caracteristică

Nr. crt.	X [mm]	Y [mm]	Z [mm]
0	-8.505	-54.150	0.452
1	-8.470	-54.200	0.446
2	-8.434	-54.249	0.441
⋮	⋮	⋮	⋮
97	-5.020	-60.149	-0.087
98	-4.984	-60.225	-0.093
99	-4.948	-60.300	-0.099

Determinarea secțiunii axiale a sculei presupune transformarea de coordonate, vezi figura 11.4:

$$\begin{aligned} X_1 &= X_{1_{i,j}} \cdot \cos\alpha + Z_{1_{i,j}} \cdot \sin\alpha; \\ Y_1 &= Y_{1_{i,j}} + a; \\ Z_1 &= -X_{1_{i,j}} \cdot \sin\alpha + Z_{1_{i,j}} \cdot \cos\alpha, \end{aligned} \tag{11.19}$$

astfel încât, sunt determinate coordonatele punctelor aparținând secțiunii axiale, vezi tabelul 11. 4 şi figura 11. 11.

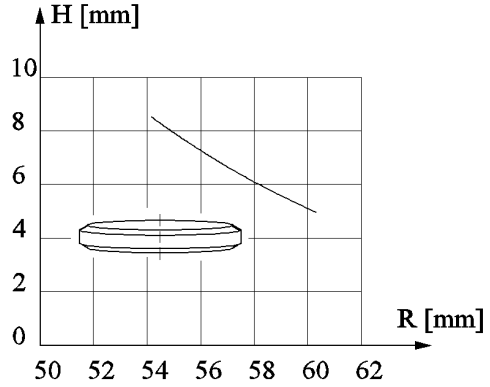


Figura 11. 11. Secțiunea axială a sculei-disc

Tabelul 11. 4.Coordonatele secțiunii axiale a sculei-disc

Nr. crt.	$H = X_1$ [mm]	$R = \sqrt{Y_1^2 + Z_1^2}$ [mm]
0	8.505	54.152
1	8.470	54.201
2	8.434	54.250
⋮	⋮	⋮
97	5.020	60.149
98	4.984	60.225
99	4.948	60.300

11.4. Profilarea sculei cilindro-frontală

În mod frecvent, se utilizează, ca sculă generaotare, scula cilindro-frontală, caracterizată de faptul că axa sa este perpendiculară pe axa suprafeței de generat, vezi figura 11. 12.

În principiu, este posibil ca axa sculei să fie suprapusă axei *Y* a sistemului de referință,

$$\vec{A} = -\vec{j} \,. \tag{11.20}$$

Vectorul de poziție al punctului $M_{i,j}$, de pe suprafața substitutivă are forma

$$\vec{r} = X_{i,j} \cdot \vec{i} + Y_{i,j} \cdot \vec{j} + Z_{i,j} \cdot \vec{k} \,, \tag{11.21}$$

cu coordonatele $X_{i,j}$, $Y_{i,j}$, $Z_{i,j}$ date de (11.2).

Prin urmare, condiția de înfăşurare devine

$$\left| \left(\vec{A}, \vec{r}, \vec{N}_{M_{i,j}} \right) \right| \le \varepsilon \tag{11.22}$$

cu $\varepsilon = \left(1 \cdot 10^{-3} \ldots 1 \cdot 10^{-2}\right)$, astfel, fiind posibilă determinarea unui ansamblu de puncte discrete pe suprafața Σ, ansamblu care va reprezenta curba caracteristică.

Ansamblul punctelor $M_{i,j}$ care satisfac condiția (11.22) reprezintă curba caracteristică a suprafeței elicoidale şi, deci, curba caracteristică a suprafeței de revoluție, care constituie suprafața periferică primară a sculei cilindro-frontale.

Principial, caracteristica C_S poate fi reprezentată în forma

$$C_S = \left\{ \begin{pmatrix} X_{i,j} \\ Y_{i,j} \\ Z_{i,j} \end{pmatrix}^T \right\}, \; (i = 1,...,n, j = 1,...,m) \,. \tag{11.23}$$

Prin rotirea curbei caracteristice (11.23) în jurul axei $\vec{A}$ (axa Y_1), cu parametrul variabil θ,

$$\begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} X_{1_{i,j}} \\ Y_{1_{i,j}} \\ Z_{1_{i,j}} \end{pmatrix}, \tag{11.24}$$

se obține suprafața periferică primară a sculei cilindro-frontale.

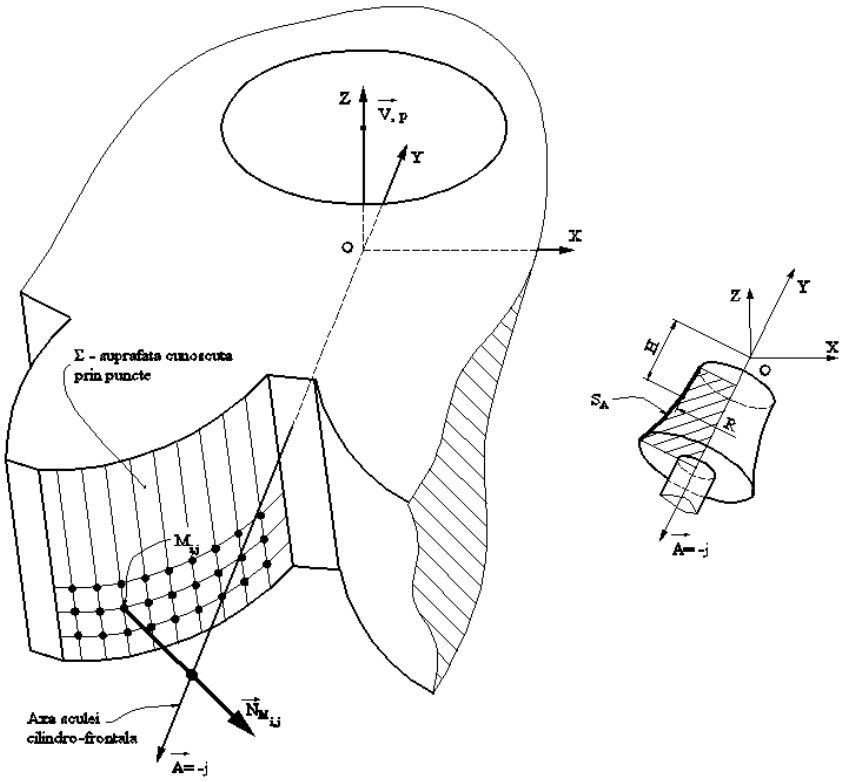


Figura 11. 12. Axa sculei cilindro-frontale

Alegerea corectă a poziției axei $\vec{A}$ față de vârtejul de suprafețe elicoidale (cazul danturilor cu dinți înclinați) permite generarea simultană a golului dintre doi dinți succesivi.

În acest caz, finalitatea problemei o constituie determinarea secțiunii axiale a frezei cilindro-frontale, vezi figura 11. 12,

$$S_A \begin{vmatrix} H = -Y_{i,j}; \\ R = \sqrt{X_{i,j}^2 + Z_{i,j}^2}. \end{vmatrix} \tag{11.25}$$

În ecuațiile (11.25), s-au notat cu $X_{i,j}$, $Y_{i,j}$, $Z_{i,j}$, coordonatele curbei caracteristice, coordonate în care punctele de pe suprafața elicoidală (11.21) îndeplinesc condiția (11.22).

11.4.1. Aproximarea punctelor pe suprafața măsurată

Se consideră suprafața măsurată a flancului suprafeței elicoidale cilindrice — flancul evolventic al roții dințate, vezi figura 11. 13.

Pe maşina de măsurat în coordonate 3D MicroHite, au fost determinate, prin măsurare directă, coordonate ale punctelor de pe generatoarele succesive ale flancului, figura 11. 13.

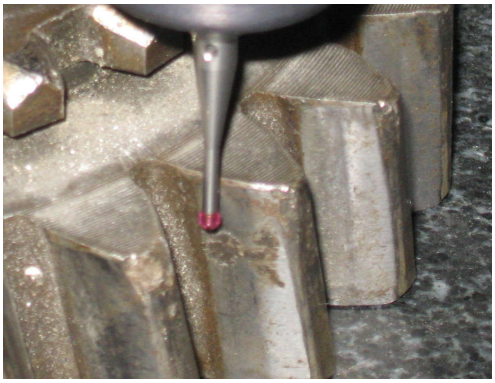


Figura 11. 13. Măsurarea roții dințate cu dinți înclinați

În tabelul 11. 5, sunt prezentate coordonate ale punctelor aparținând generatoarelor (Z=const.) ,succesive, măsurate pe suprafață .

Tabelul 11. 5. Puncte discrete măsurate pe generatoare succesive

Linia *j*	**Nr. crt.**	**X [mm]**	**Y [mm]**	**Z [mm]**
1	**1**	222.332	148.352	-450.206
	2	224.289	150.942	-450.207
	3	225.450	152.822	-450.207
	4	226.086	154.019	-450.206
	5	227.149	156.324	-450.207
⋮	⋮	⋮	⋮	⋮
5	**1**	221.968	147.720	-452.001
	2	223.809	149.664	-452.001
	3	225.382	152.003	-452.001
	4	226.496	154.047	-452.000
	5	227.328	155.798	-452.000

Ansamblul generatoarelor succesive formează suprafața discretă flancului de generat, vezi figura 11. 14.

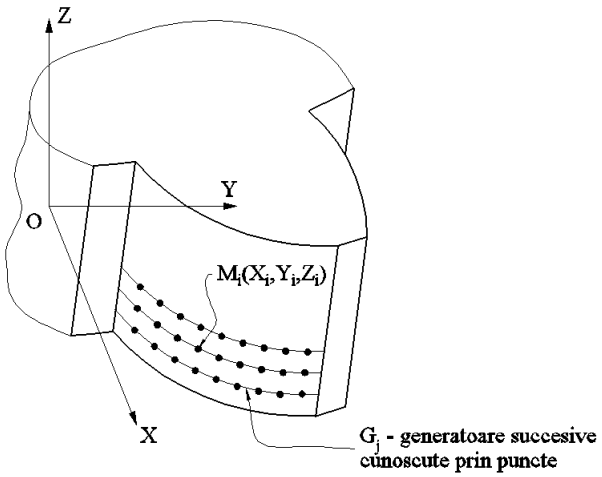


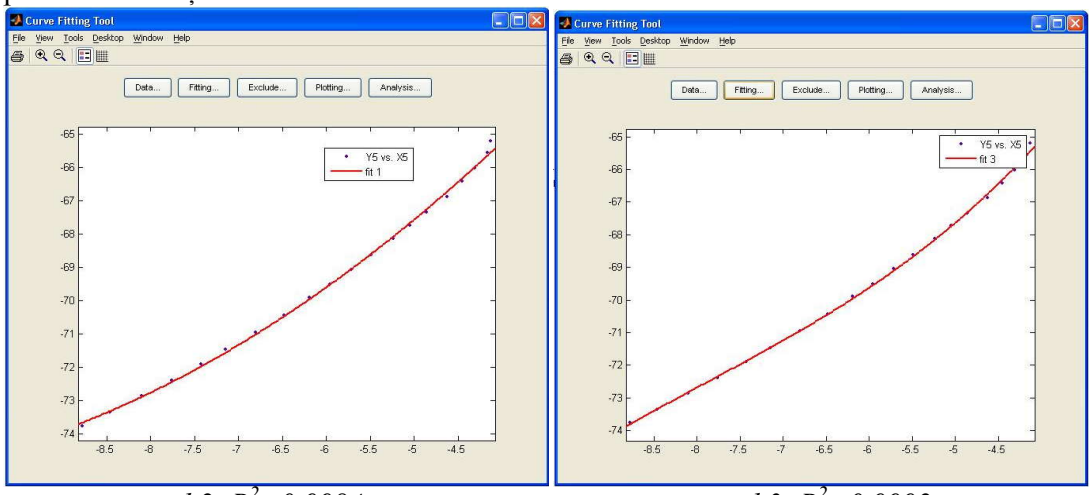
Figura 11. 14. Puncte măsurate pe flancul danturii

Este evident că, suprafața măsurată nu este o suprafață netedă fiind necesară netezirea acesteia, pentru o interpretare riguroasă a datelor măsurate.

Se propune aproximarea fiecărei generatoare printr-o ecuație polinomială care să îndeplinească următoarele condiții :

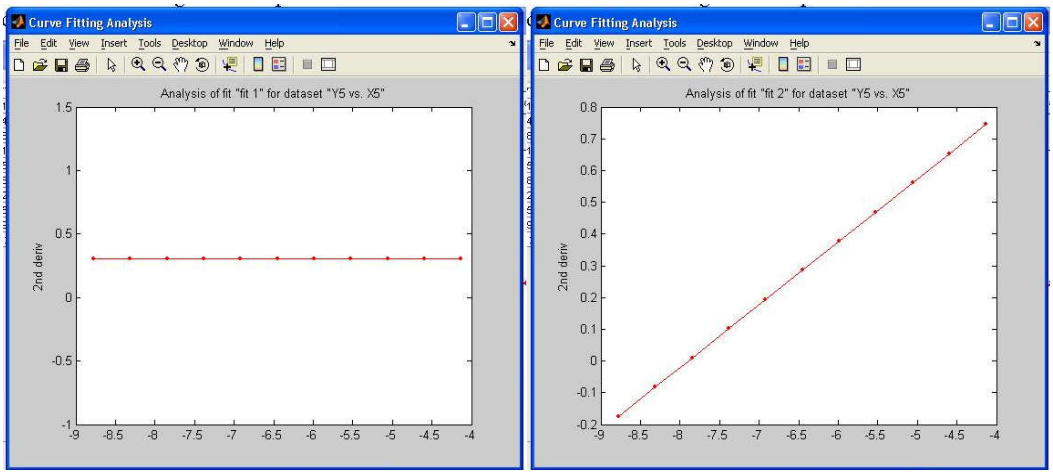
- indicele R^2 (adjusted R-square) să fie cât mai apropiat de 1;
- derivata de ordinul 2 a polinomului de substituție să fie o linie dreaptă pentru a fi evitate punctele de pe generatoar, în care există variații semnificative ale curburii.

În figura 11. 15 şi figura 11. 16, sunt prezentate formele polinoamelor de substituție pentru aproximarea datelor, precum şi derivatele de ordinul unu şi doi ale acestor polinoame, în diverse puncte ale funcției de substituire.



grad 2: R^2=0.9984; *grad 3: R^2=0.9992;*

Figura 11. 15. Forma funcției de substituire pentru diferite grade ale polinomului



gradul 2; *gradul 3;*

Figura 11. 16. Derivata de ordinul II

Din analiza formelor prezentate, se poate concluziona că indicele R^2 are cea mai apropiată valoare de 1 pentru un polinom de aproximare de ordinul 2 pentru care, în acelaşi timp, derivata de ordinul doi este liniară, fiind eliminată existența punctelor în care apar variații importante ale curburii.

Notă: Evident, pentru forme diferite ale suprafeței măsurate, polinomul de substituire va avea forme diferite.

În acest mod, toate generatoarele suprafeței au fost aproximate prin polinoame de substituire. Facem precizarea că, în funcție de coordonatele măsurate, polinoamele de substituire pot avea grade diferite.

Forma suprafeței substituite poate fi procesată cu programul Curve fitting Tools din MatLab, realizându-se o rețea mai deasă de puncte. În figura 11. 17, este prezentată o captură de ecran a suprafeței netezite.

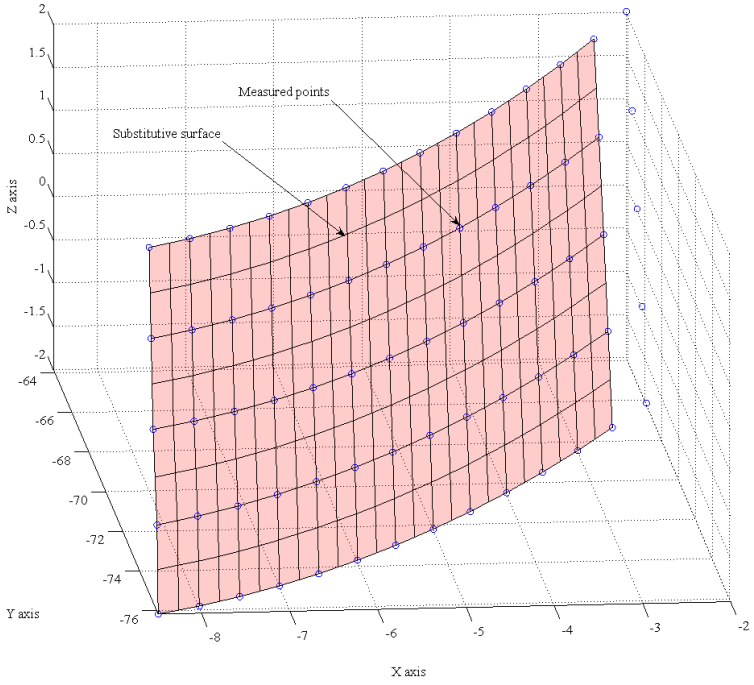


Figura 11. 17. Forma suprafeței de substituire (Curve Fitting Tools)

În tabelul 11. 6, sunt prezentate coordonatele punctelor de pe suprafața care aproximează norul de puncte efectiv măsuratsi , in figura 11.18 , forma curbei caracteristice, la generarea cu scula cilindro- frontală.

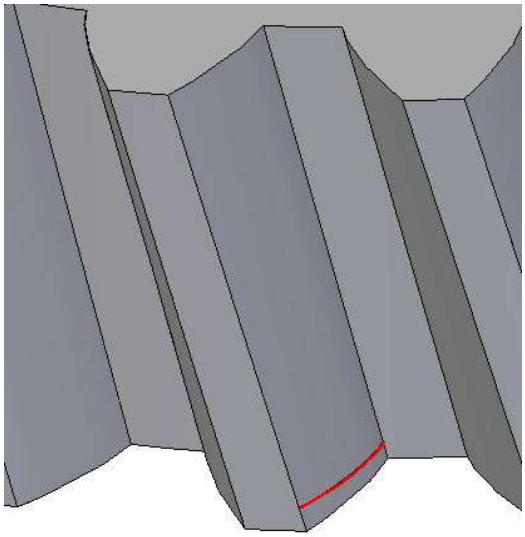


Figura 11. 18. Curba caracteristică la generarea cu scula cilindro- frontală

Tabelul 11. 6. Coordonatele punctelor generatoarelor aproximate

Linia *j*	Nr. crt.	X_1 [mm]	Y_1 [mm]	Z_1 [mm]
1	**1**	-8.407	-75.045	-2.000
	2	-6.501	-73.030	-2.000
	3	-5.082	-70.822	-2.000
	4	-3.963	-68.659	-2.000
	5	-2.955	-66.306	-2.000
⋮	⋮	⋮	⋮	⋮
5	**1**	-6.884	-74.789	2.000
	2	-5.357	-72.750	2.000
	3	-4.154	-70.796	2.000
	4	-3.332	-69.196	2.000
	5	-2.353	-67.000	2.000

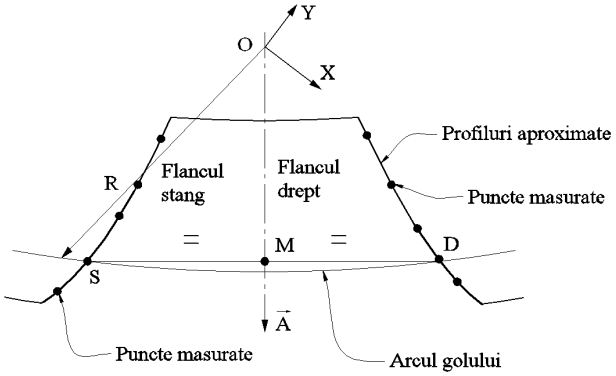


Figura 11. 19. Forma suprafeței aproximate

Scula cilindro-frontală pentru generarea golului dintre doi dinți presupune alegerea axei $\vec{A}$ ca axă de simetrie a golului dintre doi dinți succesivi. Acest lucru necesită cunoaşterea a două generatoare pe flancurile aceluiaşi gol al dinților, într-un acelaşi plan, paralel cu planul frontal, vezi figura 11. 20.

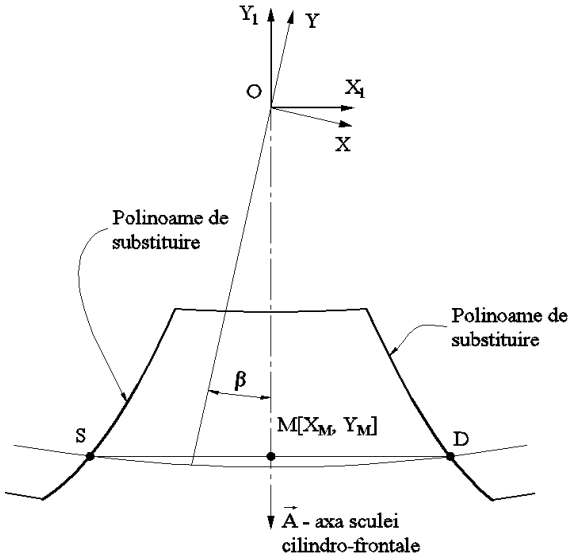


Figura 11. 20. Determinarea generatoarelor pe flancurile anti-omoloage

Pentru punctele măsurate în acelaşi plan, $Z = const.$, cele două flancuri măsurate sunt substituite cu profiluri cunoscute prin polinoame de grad superior şi, apoi, se face intersecţia acestora cu un arc de cerc având raza arbitrară. În acest fel, sunt determinate punctele *S* şi *D* pe coarda arcului $\overset{\frown}{SD}$ .Linia care uneşte punctele *O* şi *M* (punctul de mijloc al segmentului $\overline{SD}$) este axa de simetria a golului dintre doi dinţi şi poate fi considerată ca axă a sculei cilindro-frontale.

Tabelul 11. 7. Coordonate ale punctelor de pe generatoare succesive

Linia *j*	**Nr. crt.**	**X_1 [mm]**	**Y_1 [mm]**	**Z_1 [mm]**
1	**1**	-8.500	-76.179	-2.000
	2	-8.400	-76.165	-2.000
	⋮	⋮	⋮	⋮
	60	-2.600	-75.131	-2.000
	61	-2.500	-75.107	-2.000
2	**1**	-8.500	-76.107	-1.900
	2	-8.400	-76.089	-1.900
		⋮	⋮	⋮
	60	-2.600	-75.051	-1.900
	61	-2.500	-75.027	-1.900
⋮	⋮	⋮	⋮	⋮
41	**1**	-8.500	-67.358	2.000
	2	-8.400	-67.308	2.000
	⋮	⋮	⋮	⋮
	60	-2.600	-65.236	2.000
	61	-2.500	-65.175	2.000

În tabelul 11. 7, sunt prezentate coordonatele suprafeţei de substituire a norului de puncte măsurat iniţial (R_m, H_m) şi ale punctelor profilului de substituţie (R_t, H_t). Aceste coordonate au fost determinate aplicând algoritmul prezentat într-un produs informatic elaborate anterior, care permite determinarea curbei caracteristice şi a secţiunii axiale ale unei scule mărginite de suprafeţe periferice primare de rotaţie.

Neuniformitatea punctelor de pe secţiunea axială necesită netezirea curbei obţinute utilizând un polinom de aproximare de gradul 4, care va asigura o formă suficient de netedă a curbei obţinute.În acest mod, se obţin coordonatele secţiunii axiale prezentate în tabelul 11. 8.

Erorile de aproximare determinate cu instrumentul Curve Fitting din cadrul programului MatLab au următoarele valori: R-square=0.9993; Adjusted R-square=0.9991.

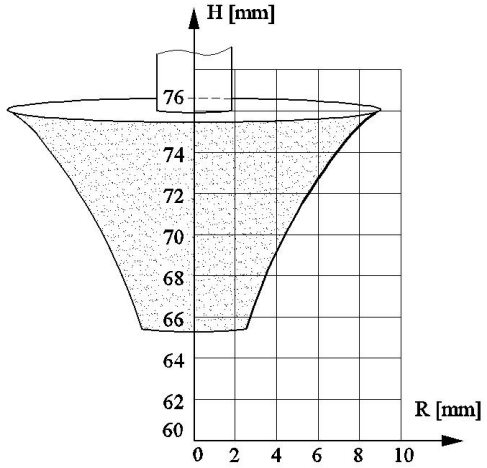


Figura 11. 21. Secțiunea axială a sculei cilindro-frontale

Tabelul 11. 8. Coordonatele punctelor pe secțiunea axială înainte de netezire

Nr. crt.	R_m [mm]	H_m [mm]	R_t [mm]	H_t [mm]
0	9.008	76.036	9.008	76.030
1	7.433	74.646	8.667	75.847
2	7.147	74.289	8.327	75.591
3	6.911	73.995	7.987	75.277
4	6.758	73.802	7.646	74.913
5	6.504	73.451	7.306	74.510
6	6.379	73.273	6.965	74.075
7	5.918	72.609	6.625	73.613
8	5.566	72.055	6.285	73.127
9	5.543	72.017	5.944	72.620
10	5.240	71.511	5.604	72.091
11	4.681	70.509	5.263	71.538
12	4.670	70.488	4.923	70.958
13	4.550	70.256	4.583	70.345
14	4.100	69.361	4.242	69.692
15	3.590	68.267	3.902	68.990
16	3.482	68.016	3.561	68.227
17	3.021	66.948	3.221	67.392
18	2.668	66.077	2.881	66.469
19	2.540	65.206	2.540	65.441

11.1 Sinteza unui produs informatic specializat

A fost dezvoltat un set de aplicații software, care acoperă o sferă largă a domeniilor de aplicabilitate ale metodologiilor propuse în proiect. Toate aplicațiile folosesc polinoame Bezier pentru a substitui profiluri teoretice sau măsurate, putând, astfel, aplica teoremele fundamentale ale gener[rii prin înfăşurare - teoreme analitice - chiar şi în cazurile profilurilor cunoscute discret (măsurate). Mai mult, algoritmul de calcul a fost în cele mai multe cazuri simplificat, aplicând *condițiile de infăsurare specifice* pentru un numar redus de puncte – punctele de control ale polinoamelor substituiente.

Aceste aplicatii au fost dezvoltate sub forma unor appleturi Java şi integrate într-o aplicație web unitară (figura 11. 22).

Aplicațiile au fost imparțite în două mari categorii:

- **2D** – grupează acele aplicații menite a proiecta scule generatoare de vârtejuri de suprafețe cilindrice, pentru a căror proiectare este suficientă analiza bidimensională (într-un plan transversal) a mișcărilor de generare; este vorba de proiectarea sculei cremalieră, a cuțitului rotativ şi a cuțitului roată.
- **3D** – grupeaza acele aplicatii referitoare la proiectarea sculelor de generare a suprafețelor elicoidale, necesitând o analiza tridimensională a mişcărilor de generare; condiția de înfăşurare este exprimată, de obicei, ca îndeplinirea coplanarității dintre axa sculei, vectorul normal la suprafața elicoidala şi vectorul ce unește centrul sculei cu punctul curent de pe suprafața elicoidală de generat. Sunt vizate: scula disc, cilindro-frontală, scula cilindrică, inelară, inelară-tangențială şi scula melc (problema specifică contactului punctiform între suparafețele în înfăşurare).

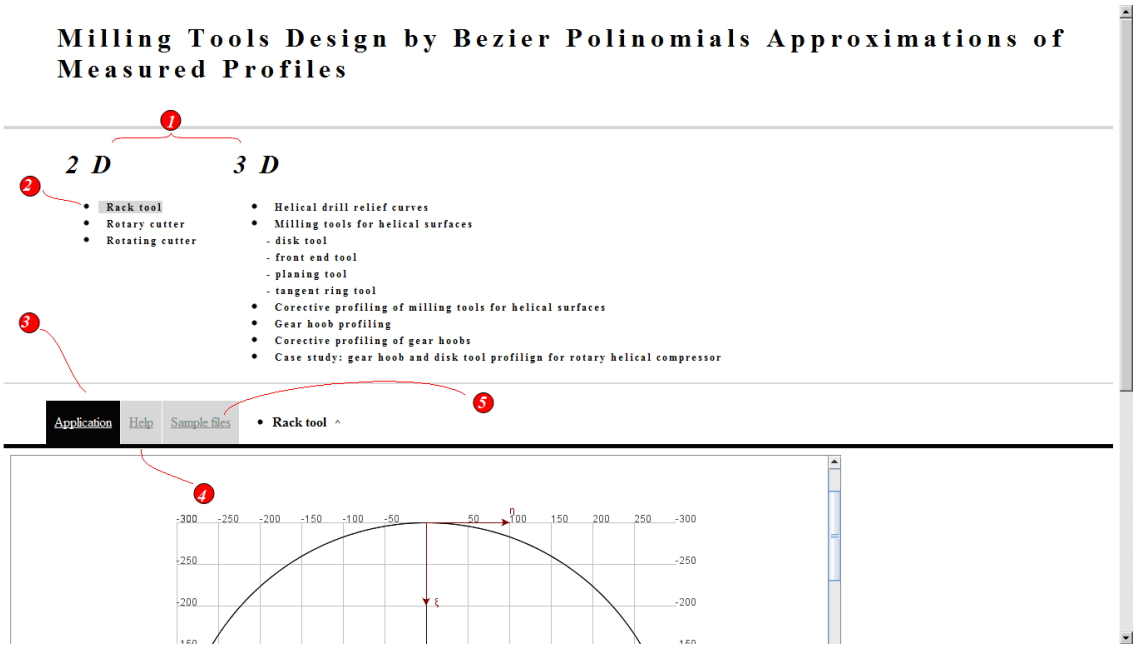


Figura 11. 22. Meniul principal

Scula cremaliera (Rack tool)

Se defineşte un ansamblu de profiluri elementare reprezentând profilul transversal al piesei de generat. Aplicația permite calculul profilului sculei cremalieră ce generează prin înfăşurare piesa. De asemenea, se calculează un profil aproximat al sculei cremalieră, utilizând polinoame Bezier. Eroarea de aproximare este determinată tot în cadrul programului.

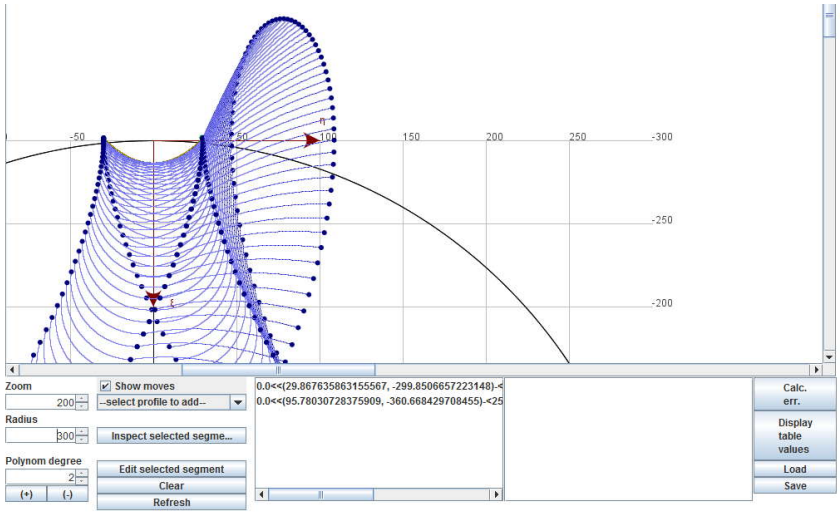


Figura 11. 23. Captura de ecran – aplicație scula cremalieră

Cuțitul roată (Rotary cutter)

Funcționalitatea este similară celei descrise descrise în secțiunea anterioara însă aplicația calculează profilul cuțitului roată care generează profilul definit (figura 11. 24)

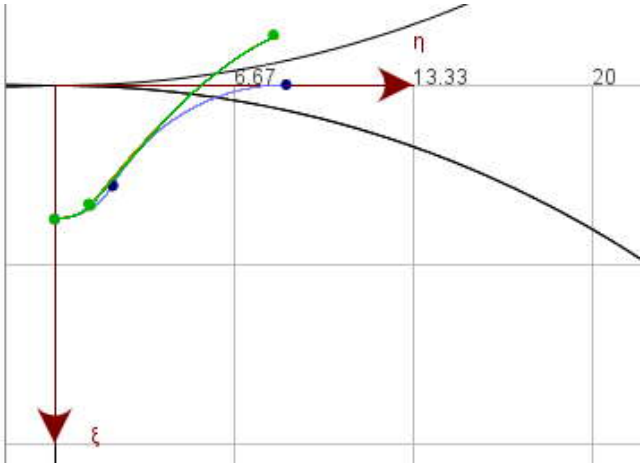


Figura 11. 24. Captura ecran – aplicație cuțitul roată

Cuțitul rotativ (Rotating cutter)

Funcționalitatea este similară celei descrise descrise în secțiunea anterioara însă aplicația calculează profilul cuțitului rotativ ce generează profilul ce reprezintă secțiunea axială unei suprefețe cilindrice elicoidale de pas constant (cazul generarii prein strunjire a melcilor, figura 11. 25)

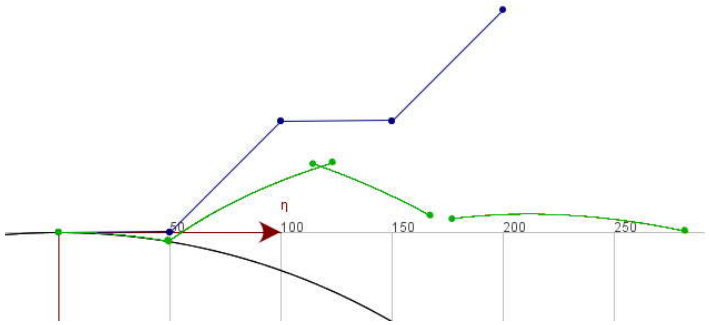


Figura 11. 25. Captura ecran – aplicație cuțitul rotativ

Curbe detalonare – ascuțire burghiu (Helical drill relief curves)

Aplicația permite introducerea unui set de puncte măsurate pe suprafața corpului abraziv de ascuțire a burghiului. Aplicația aproximează setul de puncte măsurate printr-un polinom Bezier de grad III şi trasează curbele de detalonare ale burghiului,i astfel, ascuțit.

Principalele elemente vizuale ale aplicație sunt descrise mai jos (figura 11. 26):

1 – zona în care sunt afişate curbele de detalonare ale burghiului;

2 – permite încărcarea unui fişier CSV conținand punctele măsurate ale corpului abraziv de ascuțire;

3 – numărul de puncte măsurate;

4 – valorile măsurate;

5 – calculează limitele de variație ale parametrului R0;

6 – afişează limitele parametrului R0;

7 – parametrii caracteristici ai burghiului;

8 – trasează curbele de detalonare;

9 – afişează rezultatele în formă tabelară.

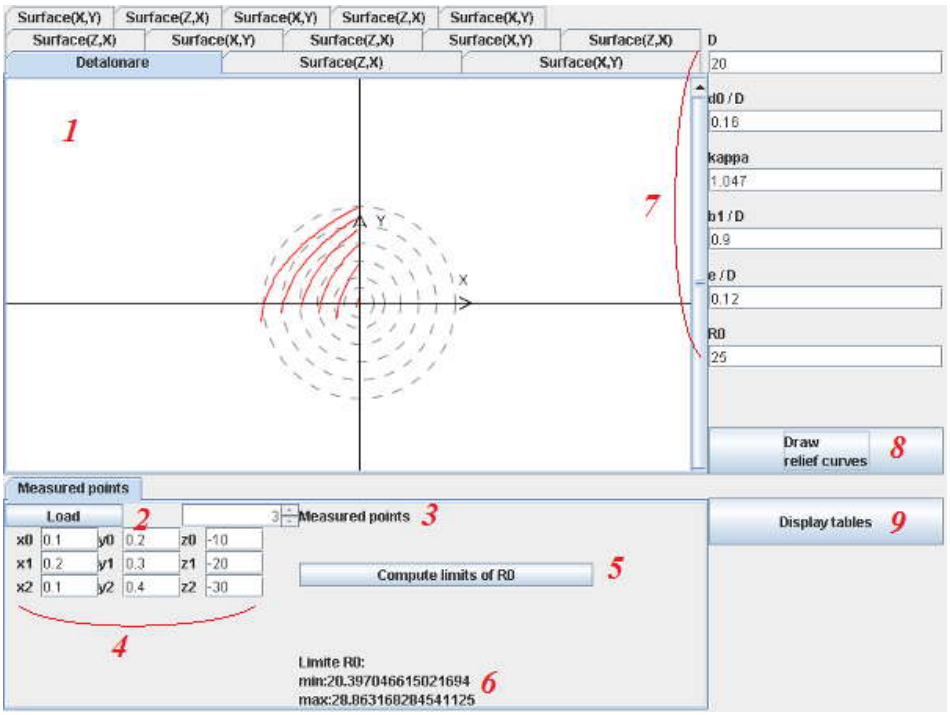


Figura 11. 26. Captura ecran – aplicație curbe de detalonare

11.5. Elaborarea unui produs informatic specializat

Scule de prelucrare a suprafețelor elicoidale (Milling tools for helical surfaces)

S-a analizat problema suparfețelor elicoidale cilindrice de pas constant cu profiluri transversale reprezentate de curbe elementare (arc de cerc, segment de dreaptă, evolventă) drept generatoare a suprafeței elicoidale ce se dorește a fi prelucrată. Aplicația determină profilul teoretic al sculei aşchetoare (scula disc, cilindro-frontală, cilindrică, inelară, inelară tangențială). Mai mult, aplicația poate calcula profilul aproximativ al sculei, folosind o aproximare prin polinoame Bezier a unui profilului teoretic dat sau a unui set de puncte măsurate. Eroarea de aproximare este calculată şi afişată. Elementele vizuale ale aplicației sunt descrise mai jos:

1 – zona principală în care sunt afişate modelele 3D ale suprafețelor elicoidale;

2 – poate fi definit un profil teoretic *arc de cerc*, ca generatoare în planul (transversal/axial), definindu-se: marimea unghiul de variație, raza şi centrul cercului corespunzător;

3 –poate fi definită *o evolventă* prin modul (m), numărul dinților (z) şi unghiul de variație;

4 –poate fi defint un *segment de dreaptă* în plan axial prin: unghi de înclinare, diametrul interior (D0) şi diametrul exterior (D) (9);

5 – se definește un *polinom de aproximare* pentru un set de puncte măsurate introduse de către utilizator sau încărcate dintr-un fişier CSV;

6 – afişează secțiunea axială a sculei;

7 – afişează proiecțiile generatoarei elicoidale;

8 – se selectează tipul sculei: disc, cilindro-frontală, cilindrică, inelară, inelară tangențială;

9 – diametrul exterior al smifabricatului

10 – unghiul de ajustare al axei sculei (doar pentru scula disc)

11 – parametrul elicoidal

12 – reconstruieşte suprafața elicoidală dupa schimbarea unor parametri

13 – calculează profilul teoretic, curba caracterisică şi eroarea de aproximație dintre profilul teoretic şi profilul aproximat, calculat, al sculei;

14 – afişează rezultate în format tabelar;

15 – controlează poziția de afişare a suprafeței elicoidale.

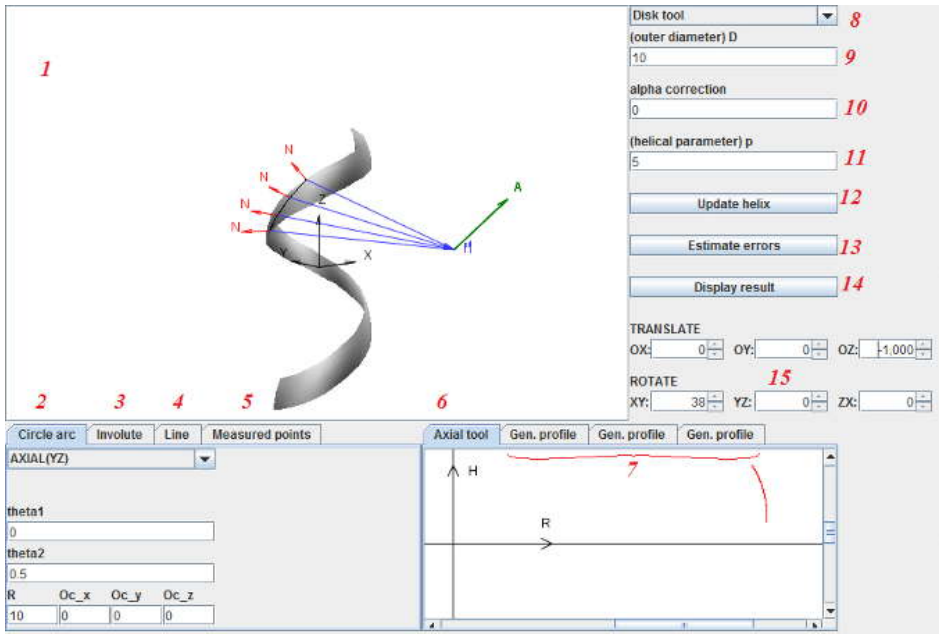


Figura 11. 27. Captura ecran – Aplicație suprafețe elicoidale:scula disc, cilindro forntală, cilindrică, scula inelară

Profilarea corectivă a sculelor de prelucrare a suprafețelor elicoidale (Milling tools for helical surfaces)

Se defineşte un profil teoretic, drept generatoarea suparafetei elicoidale, printr-o mulţime densă de puncte. De asemenea, se defineşte o mulţime de puncte măsurată, corespunzătoare profilului teoretic. Aplicaţia foloseşte un mecanism de "oglindire" pentru profilarea corectivă a sculei. Se aplică două metode de corecţie:

- se oglindesc toate punctele măsurate faţa de profilul teoretic, şi mulţimea astfel obţinută este aproximată printr-un polinom Bezier;
- se aproximează multimea de puncte măsurate cu un polinom Bezier, se oglindesc nodurile polinomului faţă de profilul teoretic, obţinându-se, astfel, un nou polinom Bezier aproximând profilul

Aplicaţia determină eroarea (diferenţa maximă) dintre profilurile axiale ale sculelor obţinute pentru cele doua profiluri oglindite (corectate).

Aplicaţia este derivată din aplicaţia descrisă in secţiunea "*Scule de prelucrare a suprafeţelor elicoidale*" iar elementele vizuale sunt asemanatoare, deosebite fiind elementele următoare (figura 11. 28):

2 – se defineşte un set dens de puncte corespunzătoare profilului teoretic, introduse de către utilizator sau încărcate dintr-un fişier CSV;

3 – se defineşte un set dens de puncte măsurate introduse de către utilizator sau încărcate dintr-un fişier CSV;

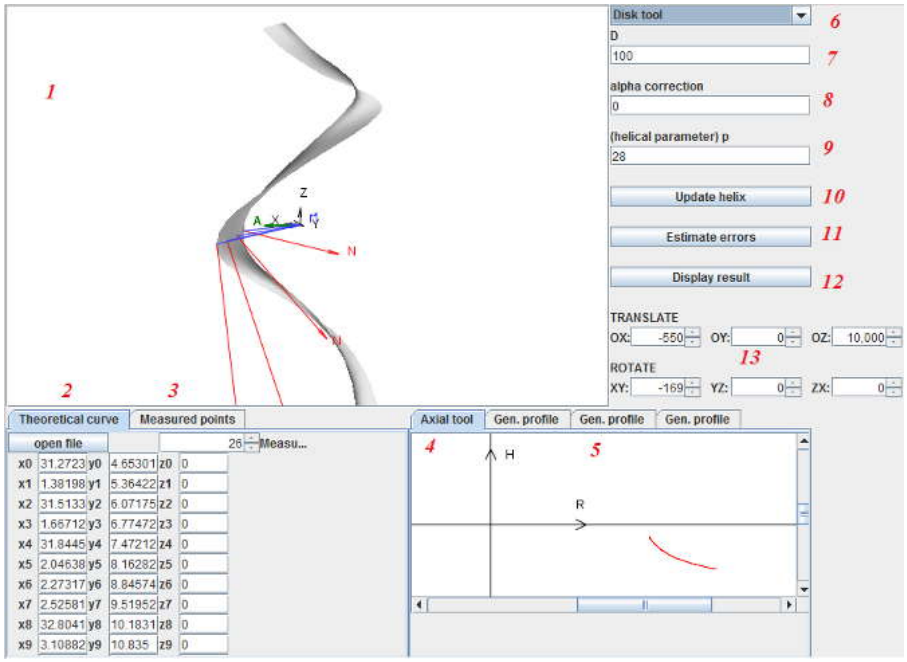


Figura 11. 28. Captură ecran – aplcaţie profilarea corectivă – scule suprafeţe elicoidale

Profilarea sculei melc

Aplicaţia este o extindere a aplicaţiei descrisă în secţiunea "*Scule de prelucrare a suprafeţelor elicoidale*". Se defineşte profilul semifabricatului ca un profil compus din profiluri elementare (profilul piesei de generat). Aplicaţia determină profilul cremalierei ce generează piesa şi, mai mult, generează profilul sculei melc înfăşuratoare cremalierei. Algoritmul de calcul este, de asemenea, bazat pe polinoame Bezier. Elementele vizuale ale aplicaţiei (figura 11. 29) diferă faţă de cele prezentate în secţiunea "*Scule de prelucrare a suprafeţelor elicoidale*" prin:

14 – se afişează o nouă fereastră pentru a configura diverşi parametrii ai sculei melc(figura 11. 30) ;

18 - modelul suprafaţei periferice primare a sculei melc;

19 – raza sculei (mm);

20 – unghiul de înclinare al sculei melc;

21 – parametrul elicoidal (mm);

22 – pasul circular al vîrtejului de suprafeţe ale piesei (mm) ;

23 – redesenează suprafaţa sculei melc după modificarea parametrilor;

24 – calculează profilul teoretic, curba caracteristică şi eroarea de aproximare dintre secţiunea teoretică a sculei melc şi cea calculată prin aproximări cu polinoame Bezier ;

25 – afişează rezultatele în formă tabelară;

26 – controlează poziţia suprafeţei;

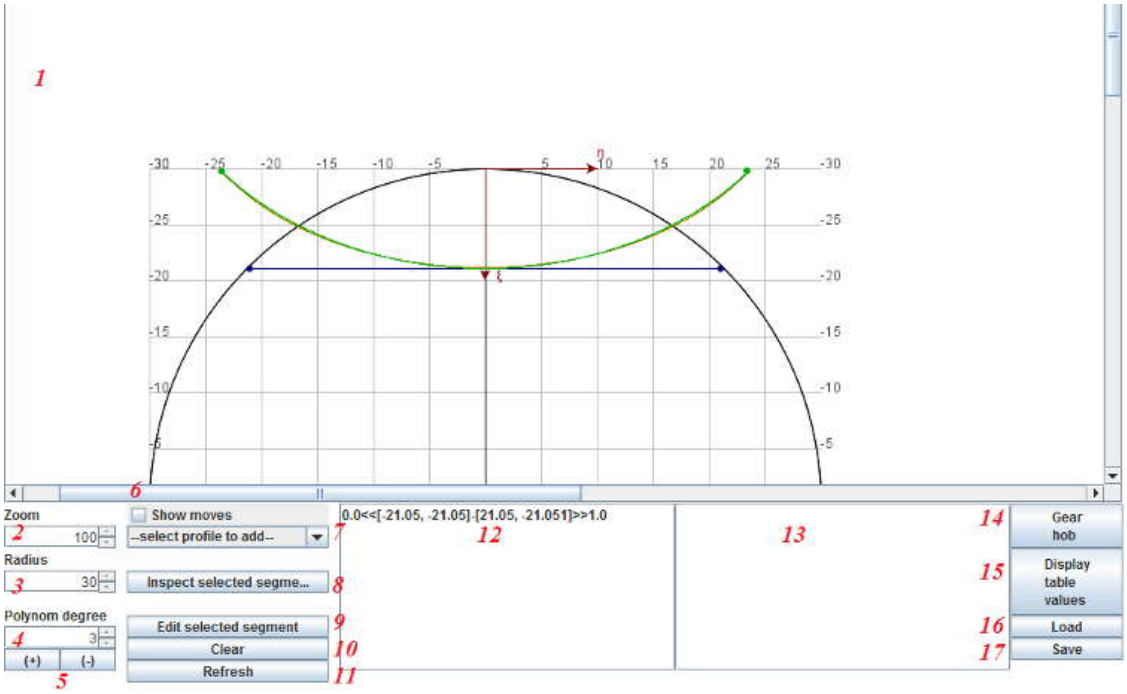


Figura 11. 29. Captura ecran aplicaţie scula melc – profilul transversal al piesei şi al cremalierei

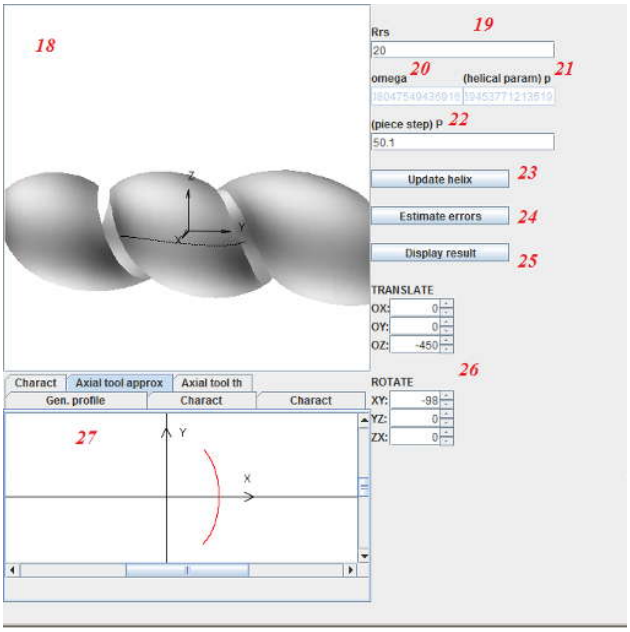


Figura 11. 30. Suprafaţa periferică primară a sculei melc

Profilarea corectivă a sculei melc

Aplicaţia combină o serie de elemente deja prezentate în aplicaţiile anterioare (secţiunile "*Profilarea corectivă a scullore de prelucrare a suprafeţelor elicoidale*", "*Profilarea sculei melc*"). Se defineşte un profil măsurat şi un profil teoretic al piesei. Aplicaţia determină "oglindirea"profilului măsurat şi, pe baza acestui noi profil oglindi, se determină profilul sculei cremaliere şi a sculei melc corespunzătoare (profilul corectat al sculei)

Studiu de caz: profilarea sculei melc şi a sculei disc pentru generarea rotoarelor compresoarelor elicoidale

Pornind de la profilul cremalierei reciproc înfăşurătoare secţiunilor transversale ale rotoarelor compresorului elicoidal, aplicaţia determină profilurile sculei melc şi ale sculei disc pentru generarea rotoarelor. Profilul cremalierei este compus din profiluri elementare (arce de cerc şi segmente de dreaptă) şi prezintă o discontinuitate. Discontinuitatea este suplinită prin două polinoame Bezier de gradul II (figura 11. 31) .

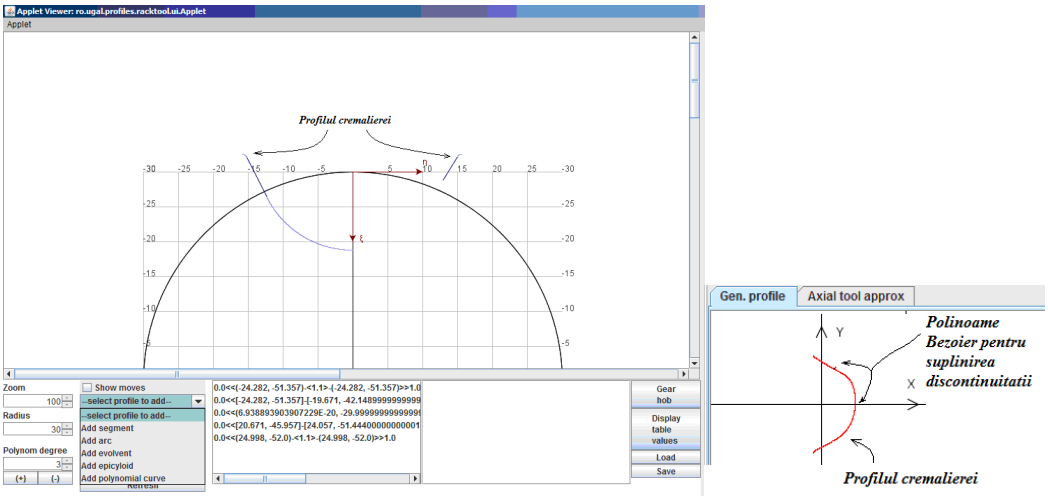


Figura 11. 31. Profilul cremalierei

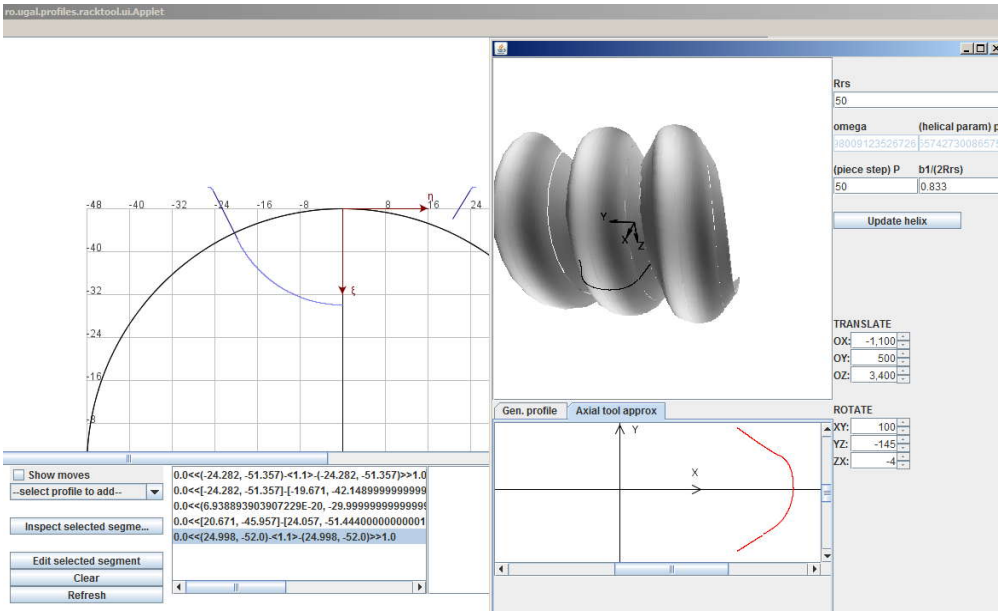


Figura 11. 32. Scula melc

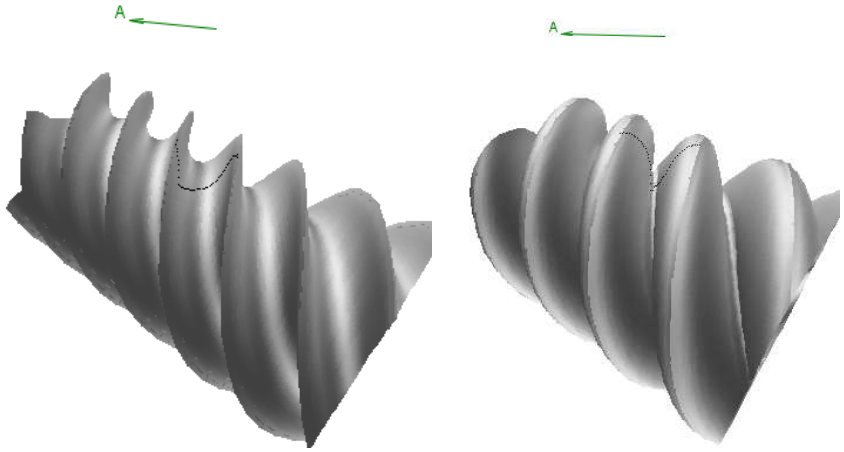
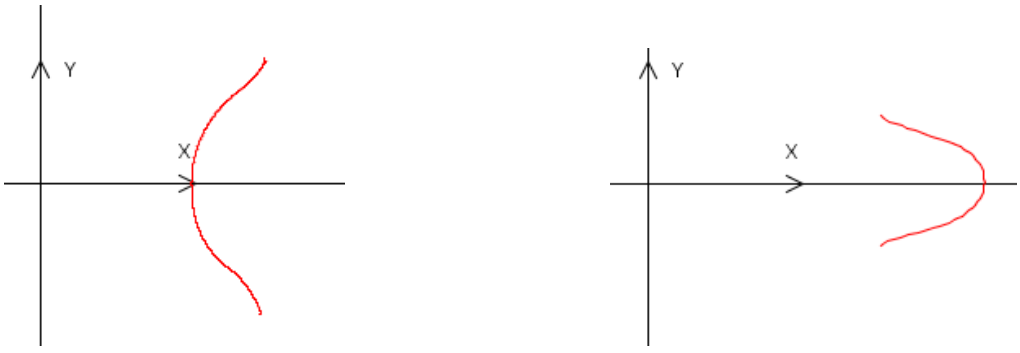


Figura 11. 33. Rotorii compresorului



a). melc condus *b). scula disc*

Figura 11. 34. Profilul melcului condus şi al sculei disc

a). melc conducator *b). scula disc*

Figura 11. 35. Profilul melcului conducător şi al sculei disc

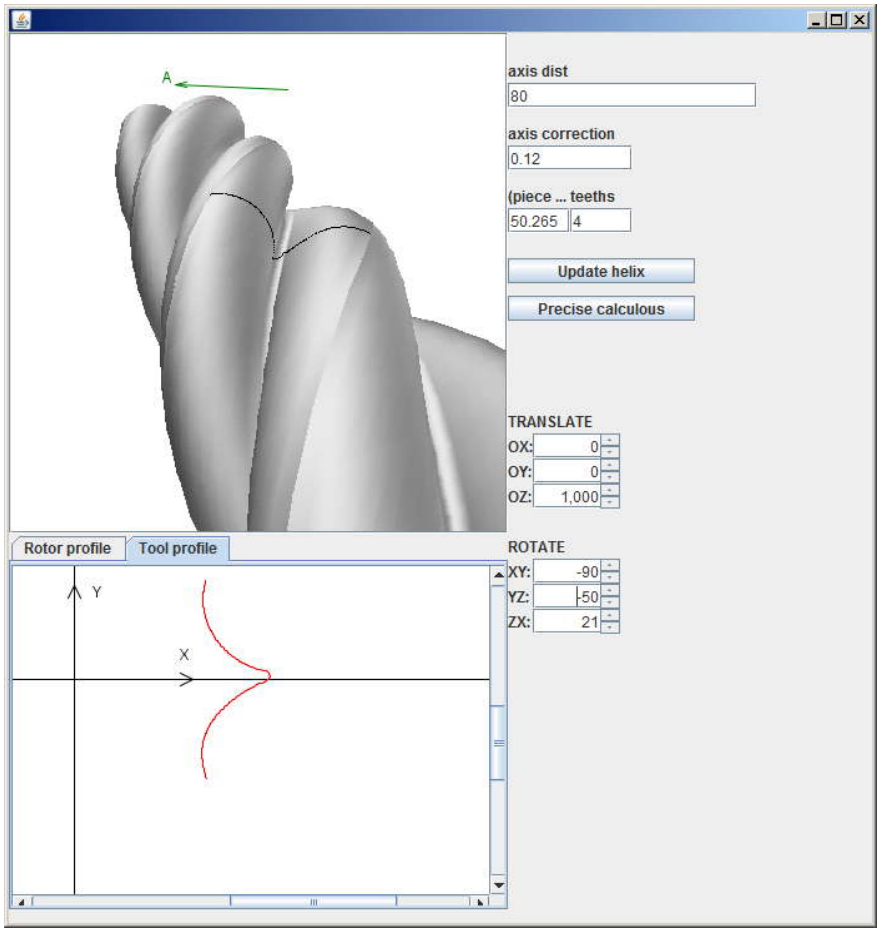


Figura 11. 36. Profilarea sculei disc pentru generarea rotorilor

In figurile 11.32-11.36, sunt prezentate elemetele generate cu programul propus referitoare la profilarea sculei melc şi profilarea de corecție a acestei scule. Totodată, sunt prezentate profiluri ale sculelor de ordinul doi, determinate in baza programului prezentat, pentru profilarea sculei disc generatoare a melcilor componenți ai compresorului elicoidal.

11.6. Raport final

A fost elaborate raportul general al proiectului ID_656/2007, cuprinzând obiectivele şi activitățile desfăşurate pe întreaga perioadă a proiectului (2007-2010).

Acest raport general cuprinde:

EXTRASE DIN CODUL SURSĂ

```
package ro.ugal.model;
import java.awt.geom.Point2D;

public class PolynomialCurve {
    private int degree;
    private double[] coeficientsX;
    private double[] coeficientsY;
    private double[] coeficientsZ;

    private double[] lambda = null;

    public PolynomialCurve(int degree) {
        this.degree = degree;
    }

    public void setCoeficients(
            double[] coeficientsX,
            double[] coeficientsY,
            double[] coeficientsZ) {
        this.coeficientsX = coeficientsX;
        this.coeficientsY = coeficientsY;
        this.coeficientsZ = coeficientsZ;
    }

    public int getDegree() {
        return degree;
    }

    public void setDegree(int degree) {
        this.degree = degree;
    }

    public Point3D getPointAt(double lambda) {
        double xx = 0;
        double yy = 0;
        double zz = 0;
        for (int j = 0; j <= degree; j++) {
            xx += coeficientsX[j] * Math.pow(lambda, j) * Math.pow(1 - lambda, degree - j);
            yy += coeficientsY[j] * Math.pow(lambda, j) * Math.pow(1 - lambda, degree - j);
            zz += coeficientsZ[j] * Math.pow(lambda, j) * Math.pow(1 - lambda, degree - j);
        }
        return new Point3D(xx, yy, zz);
    }

    /**
     * (A * B)' = A' * B + A * B'
     * P(x)  = sum [ Cj * x^j * (1 - x)^(D - j) ] <br>
     * P'(x) = sum [ Cj * (j * x^(j-1) * x- (D-j) * x^(D-j-1) ] = sum [ Cj * (j * x^(j-1) - D
* x^(D-1)) ]
     */
    public Point3D getDerivativePointAt(double lambda) {
        double DX =
                coeficientsX[3] * 3 * lambda * lambda +
                coeficientsX[2] * (2 * lambda * (1 - lambda) - lambda * lambda) +
                coeficientsX[1] * ((1 - lambda) * (1 - lambda) - 2 * lambda * (1 - lambda)) -
                coeficientsX[0] * (3 * (1 - lambda) * (1 - lambda));
        double DY =
                coeficientsY[3] * 3 * lambda * lambda +
                coeficientsY[2] * (2 * lambda * (1 - lambda) - lambda * lambda) +
                coeficientsY[1] * ((1 - lambda) * (1 - lambda) - 2 * lambda * (1 - lambda)) -
                coeficientsY[0] * (3 * (1 - lambda) * (1 - lambda));
        double DZ =
                coeficientsZ[3] * 3 * lambda * lambda +
                coeficientsZ[2] * (2 * lambda * (1 - lambda) - lambda * lambda) +
                coeficientsZ[1] * ((1 - lambda) * (1 - lambda) - 2 * lambda * (1 - lambda)) -
                coeficientsZ[0] * (3 * (1 - lambda) * (1 - lambda));
        return new Point3D(DX, DY, DZ);

    }

    public void setLambda(double[] lambda) {
        this.lambda = lambda;
    }
```

```
    public double[] getLambda() {
        return lambda;
    }

    public DiscreteCurve getAsDiscretecurve() {
        DiscreteCurve discreteCurve = new DiscreteCurve();
        for (double lambda = 0; lambda <= 1; lambda += 0.01) {
            discreteCurve.addPoint(this.getPointAt(lambda));
        }
        return discreteCurve;
    }
}
```

```
package ro.ugal.approximation;

import ro.ugal.model.DiscreteCurve;
import ro.ugal.model.PolynomialCurve;
import ro.ugal.model.Point3D;
import java.awt.geom.Point2D;
import java.text.DecimalFormat;
import org.jscience.mathematics.vector.Float64Matrix;
import org.jscience.mathematics.vector.Matrix;
import org.jscience.mathematics.vector.Float64Vector;
import org.jscience.mathematics.vector.DenseVector;
import org.jscience.mathematics.number.Float64;

public class PolynomialApproximator {
    public static String report;

    public static PolynomialCurve getApproximation(
            DiscreteCurve discreteCurve,
            int degree) {

        PolynomialCurve polinomial = new PolynomialCurve(degree);
        int points = discreteCurve.getNumberOfPoints();

        double totalDist = 0;
        for (int i = 1; i < points; i++) {
            Point3D p1 = discreteCurve.getPointAt(i);
            Point3D p2 = discreteCurve.getPointAt(i - 1);
            totalDist += p1.distance(p2);
        }

        double[] lambda = new double[(int) degree + 1];

        lambda[0] = 0;
        lambda[degree] = 1;
        double cumultativeDist = 0;

        int index = 1;
        int indices[] = new int[degree + 1];
        indices[0] = 0;
        indices[degree] = points - 1;
        for (int i = 0; i < points - 1; i++) {
            Point3D p1 = i > 0 ? discreteCurve.getPointAt(i - 1) : null;
            Point3D p2 = discreteCurve.getPointAt(i);
            Point3D p3 = discreteCurve.getPointAt(i + 1);

            cumultativeDist += p1 != null ? p1.distance(p2) : 0;

            double fraction = totalDist * index / degree;

            double p23Dist = p2.distance(p3);

            if (    (cumultativeDist  <= fraction) &&
                    (cumultativeDist + p23Dist >= fraction)) {

                if (Math.abs(cumultativeDist - fraction) <= Math.abs(cumultativeDist + p23Dist
- fraction)) {
                    lambda[index] = cumultativeDist / totalDist;
                    indices[index] = i;
                    index++;
                } else {
                    lambda[index] = (cumultativeDist + p23Dist) / totalDist;
                    indices[index] = (i + 1);
```

```
            index++;
        }
    }
}
if (discreteCurve.getNumberOfPoints() == degree + 1) {
    cumultativeDist = 0;
    for (int i = 1; i <= degree; i++) {
        indices[i] = i;
        Point3D p1 = discreteCurve.getPointAt(i);
        Point3D p2 = discreteCurve.getPointAt(i - 1);
        cumultativeDist += p1.distance(p2);
        lambda[i] = cumultativeDist / totalDist;
    }
}

for (int i = 0; i <= degree; i++) {
    System.out.println("LAMBDA[" + i + "]: >" + lambda[i]);
    System.out.println("INDICES[" + i + "]: >" + indices[i]);
}
polinomial.setLambda(lambda);
double[][] matrix = new double[(int) degree + 1][(int) degree + 1];
for (int i = 0; i <= degree; i++) {
    for (int j = 0; j <= degree; j++) {
        matrix[i][j] = Math.pow(lambda[i], j) * Math.pow(1 - lambda[i], degree - j);
    }
}
double[]  coeficientsY = new double[(int)degree + 1];
double[]  coeficientsX = new double[(int)degree + 1];
double[]  coeficientsZ = new double[(int)degree + 1];

{
    double[] values = new double[(int) degree + 1];
    for (int i = 0; i <= degree; i++) {
        values[i] = discreteCurve.getPointAtRelativePosition(lambda[i]).x;
    }
    Float64Matrix M = Float64Matrix.valueOf(matrix);
    Matrix result = M.solve(
            Float64Matrix.valueOf(
                    new Float64Vector[]{
                            Float64Vector.valueOf(values)
                    }
            ).transpose()
    );

    //result for coeficientsX
    DenseVector vector1 = (DenseVector) result.getColumn(0);

    for (int i = 0; i < degree + 1; i++) {
        coeficientsX[i] = ((Float64) vector1.get(i)).doubleValue();
    }
}

{
    double[] values = new double[(int) degree + 1];
    for (int i = 0; i <= degree; i++) {
        values[i] = discreteCurve.getPointAtRelativePosition(lambda[i]).y;
    }
    Float64Matrix M = Float64Matrix.valueOf(matrix);
    Matrix result = M.solve(
            Float64Matrix.valueOf(
                    new Float64Vector[]{
                            Float64Vector.valueOf(values)
                    }
            ).transpose()
    );

    //result for coeficientsY
    DenseVector vector1 = (DenseVector) result.getColumn(0);

    for (int i = 0; i < degree + 1; i++) {
        coeficientsY[i] = ((Float64) vector1.get(i)).doubleValue();
    }
}

{
    double[] values = new double[(int) degree + 1];
    for (int i = 0; i <= degree; i++) {
```

```
            values[i] = discreteCurve.getPointAtRelativePosition(lambda[i]).z;
        }
        Float64Matrix M = Float64Matrix.valueOf(matrix);
        Matrix result = M.solve(
                Float64Matrix.valueOf(
                        new Float64Vector[]{
                                Float64Vector.valueOf(values)
                        }
                ).transpose()
        );

        //result for coeficientsZ
        DenseVector vector1 = (DenseVector) result.getColumn(0);

        for (int i = 0; i < degree + 1; i++) {
            coeficientsZ[i] = ((Float64) vector1.get(i)).doubleValue();
        }
    }

    for (int i = 0; i < degree + 1; i++) {
        System.out.println("COEFICEINTS_X[" + i + "]=" + coeficientsX[i]);
    }

    for (int i = 0; i < degree + 1; i++) {
        System.out.println("COEFICEINTS_Y[" + i + "]=" + coeficientsY[i]);
    }

    for (int i = 0; i < degree + 1; i++) {
        System.out.println("COEFICEINTS_Z[" + i + "]=" + coeficientsZ[i]);
    }

    double[] maxError = new double[points + 1];
    for (int i = 0; i < points; i++) {
        maxError[i] = 10000;
    }

    for (double l = 0; l <= 1; l += 0.005) {
        double xx = 0;
        double yy = 0;
        double zz = 0;
        for (int j = 0; j <= degree; j++) {
            xx += coeficientsX[j] * Math.pow(l, j) * Math.pow(1 - l, degree - j);
            yy += coeficientsY[j] * Math.pow(l, j) * Math.pow(1 - l, degree - j);
            zz += coeficientsZ[j] * Math.pow(l, j) * Math.pow(1 - l, degree - j);
        }
        for (int i = 0; i < points; i++) {
            double dist =discreteCurve.getPointAt(i).distance(new Point3D(xx, yy, zz));
            if (dist < maxError[i]) {
                maxError[i] = dist;
            }
        }
    }

    double maxErrorTotal = maxError[0];
    for (int i = 0; i < points; i++) {
        if (maxErrorTotal < maxError[i]) {
            maxErrorTotal = maxError[i];
        }
    }
    System.out.println("MAX ERROR:" + maxErrorTotal);
    polinomial.setCoeficients(coeficientsX, coeficientsY, coeficientsZ);
    return polinomial;
}

public static double maxError(
        DiscreteCurve   referenceCurve,
        DiscreteCurve approximatedCurve) {
    report = "";
    double maxError = 0;
    double lambdaMaxError = 0;
    StringBuffer reportSB = new StringBuffer();
    reportSB.append("<table>");
    reportSB.append("<tr>");
    reportSB.append("<td>lambda</td>");
    reportSB.append("<td>Approx.x</td>");
    reportSB.append("<td>Approx.y</td>");
    reportSB.append("<td>Th.x</td>");
```

```
            reportSB.append("<td>Th.y</td>");
            reportSB.append("</tr>");
            int index = 0;
            for (int lambdaInt = 0; lambdaInt <= 1000; lambdaInt += 1) {
                double lambda1 = ((double) lambdaInt) / (1000);

                Point3D point = referenceCurve.getPointAtRelativePosition(lambda1);
                double xx = point.x;
                double yy = point.y;
                double xMindist = 0;
                double yMindist = 0;
                double minDist = 100000;
                for (double lambda2 = (lambda1 - 0.1 >= 0 ? lambda1 - 0.1 : 0);
                            lambda2 <= (lambda1 + 0.1 <= 1 ? lambda1 + 0.1 : 1);
                            lambda2 += 0.001) {
                    Point3D pointRef = approximatedCurve.getPointAtRelativePosition(lambda2);
                    double dist = Math.sqrt(
                            (xx - pointRef.x) * (xx - pointRef.x)
                            +
                            (yy - pointRef.y) * (yy - pointRef.y)
                    );
                    if (dist < minDist) {
                        minDist = dist;
                        xMindist = pointRef.x;
                        yMindist = pointRef.y;
                    }
                }

                DecimalFormat f = new DecimalFormat("0.000");


                boolean displayBoldValues = false;
                int degree = 3;
                if (degree == 2 && (lambdaInt == 0 || lambdaInt == 50 || lambdaInt == 100)) {
                    displayBoldValues = true;
                }
                if (degree == 3 && (lambdaInt == 0 || lambdaInt == 333 || lambdaInt == 666 ||
lambdaInt == 1000)) {
                    displayBoldValues = true;
                }
                if (degree == 4 && (lambdaInt == 0 || lambdaInt == 250 || lambdaInt == 500 ||
lambdaInt == 750 || lambdaInt == 1000)) {
                    displayBoldValues = true;
                }

                if (index % 50 == 0 || displayBoldValues) {
                    if (displayBoldValues) {
                        reportSB.append(
                                "<tr><td><b>" + f.format(lambda1) + "</td>" +
                                "<td><b>" + f.format(xx) + "</b></td>" +
                                "<td><b>" + f.format(yy) + "</b></td>" +
                                "<td><b>" + f.format(xMindist) + "</b></td>" +
                                "<td><b>" + f.format(yMindist) + "</b></td>" +
                                "<td><b>" + f.format(100 * minDist) + "E-2</b></td></tr>");
                    } else {
                        reportSB.append(
                                "<tr><td>" + f.format(lambda1) + "</td>" +
                                "<td>" + f.format(xx) + "</td>" +
                                "<td>" + f.format(yy) + "</td>" +
                                "<td>" + f.format(xMindist) + "</td>" +
                                "<td>" + f.format(yMindist) + "</td>" +
                                "<td>" + f.format(100 * minDist) + "E-2</td></tr>");
                    }
                }
                index++;
                if (minDist > maxError) {
                    maxError = minDist;
                    lambdaMaxError = lambda1;
                }
            }
            reportSB.append("</table>");
            reportSB.append("LAMBDA MAX ERROR:" + lambdaMaxError + "<br/>");
            reportSB.append("MAX ERROR:" + maxError + "<br/");
            report = reportSB.toString();
            return maxError;
        }
    }
```

```
package ro.ugal.model;

public class MathUtils {

    public static double determinant3X3(
            double a, double b, double c,
            double d, double e, double f,
            double g, double h, double i
            ) {
        return a*e*i - a*f*h - b*d*i + b*f*g + c*d*h - c*e*g;
    }

    /**
     * A00 A10 A20      B0    A00 *B0 + A10 * B1 + A20 * B2
     * A01 A11 A21      B1 =  ...
     * A02 A12 A22      B2    ...
     *
     * @param A
     * @param B
     * @return A x B
     */
    public static double[] multiply3X3with3(double[][] A, double[] B) {
        return new double[] {
                B[0] * A[0][0] + B[1] * A[0][1] + B[2] * A[0][2],
                B[0] * A[1][0] + B[1] * A[1][1] + B[2] * A[1][2],
                B[0] * A[2][0] + B[1] * A[2][1] + B[2] * A[2][2]
        };
    }

    public static double coplanarity(
            double x1, double y1, double z1,
            double x2, double y2, double z2,
            double x3, double y3, double z3,
            double x4, double y4, double z4) {
        double ret =
                - determinant3X3(
                    x2, y2, z2,
                    x3, y3, z3,
                    x4, y4, z4
                ) + determinant3X3(
                    x1, y1, z1,
                    x3, y3, z3,
                    x4, y4, z4
                ) - determinant3X3(
                    x1, y1, z1,
                    x2, y2, z2,
                    x4, y4, z4
                ) + determinant3X3(
                    x1, y1, z1,
                    x2, y2, z2,
                    x3, y3, z3
                );

        return ret;
    }
}
```

```
package ro.ugal.helix.tool;

import ro.ugal.model.Point3D;

import java.awt.geom.Point2D;

public interface Tool {

    Point3D getToolOrigin(Point3D currentPoint);

    Point3D getToolAxis();

    double getAlpha();

    double envelopingCondition(Point3D normal, Point3D currentPoint);
```

```
        Point3D changeCoordinates(Point3D currentPoint, double step);

        Point2D.Double computeSecondOrderProfile(Point3D currentPointInToolSystem);
    }
```

```
    package ro.ugal.helix.tool.impl;

    import ro.ugal.helix.tool.Tool;
    import ro.ugal.model.Point3D;
    import ro.ugal.model.MathUtils;

    import java.awt.geom.Point2D;

    public class DiskTool implements Tool {
        public double alpha = 0;

        //distanta de la originea sculei la originea burghiului
        public double a = 50d;

        /**
         * Creates a new disk tool
         * @param P - step
         * @param d1 - outer diameter
         */
        public DiskTool(double P, double d1, double alphaCorection) {
            a = d1 * 5;
            alpha = alphaCorection + Math.atan(2 * P / d1);

            System.out.println("ALPHA:" + alpha);
            System.out.println("A:" + a);
        }

        public Point3D getToolOrigin(Point3D currentPoint) {
            return new Point3D(a, 0 , 0);
        }

        public Point3D getToolAxis() {
            //axa sculei
            return new Point3D(a, -20 * Math.sin(alpha), 20 * Math.cos(alpha));

        }

        public double getAlpha() {
            return alpha;
        }

        public double envelopingCondition(Point3D normal, Point3D currentPoint) {
            return MathUtils.coplanarity(
                normal.x + currentPoint.x , normal.y + currentPoint.y , normal.z + currentPoint.z,
                getToolAxis().x, getToolAxis().y, getToolAxis().z,
                //a, -Math.sin(alpha), Math.cos(alpha),
                currentPoint.x, currentPoint.y, currentPoint.z,
                //a, 0 , 0
                getToolOrigin(currentPoint).x,                      getToolOrigin(currentPoint).y,
getToolOrigin(currentPoint).z
            );
        }

        public Point3D changeCoordinates(Point3D currentPoint, double step) {
            double[] xyz = MathUtils.multiply3X3with3(
                    new double[][]{
                            {1, 0                    , 0                    },
                            {0, Math.cos(getAlpha()) , Math.sin(getAlpha()) },
                            {0, -Math.sin(getAlpha()), Math.cos(getAlpha()) },
                    },
                    new double[] {
                            currentPoint.x - a, currentPoint.y, currentPoint.z
                    }
            );

            return new Point3D(
                    xyz[0],
                    xyz[1],
                    xyz[2]
            );
        }
```

```
        public Point2D.Double computeSecondOrderProfile(Point3D currentPointInToolSystem) {
            double h = currentPointInToolSystem.z;
            double r = Math.sqrt(
                    currentPointInToolSystem.x * currentPointInToolSystem.x +
                    currentPointInToolSystem.y * currentPointInToolSystem.y
            );
            return new Point2D.Double(r, h);
        }
    }
```

```
    package ro.ugal.helix.tool.impl;

    import ro.ugal.model.Point3D;
    import ro.ugal.model.MathUtils;
    import ro.ugal.helix.tool.Tool;

    import java.awt.geom.Point2D;

    /**
     * Date: Sep 5, 2008
     * Time: 5:59:39 PM
     */
    public class FrontalCylinderTool implements Tool {
        public static final double a = 0;

        private double alpha = 0;

       public Point3D getToolOrigin(Point3D currentPoint) {
            return new Point3D(0, 0 , 0);
        }

        public Point3D getToolAxis() {
            //axa sculei
            return new Point3D(20, 0, 0);
        }

        public double getAlpha() {
            return 0;
        }

        public double envelopingCondition(Point3D normal, Point3D currentPoint) {
            return MathUtils.coplanarity(
                normal.x, normal.y, normal.z,
                getToolAxis().x, getToolAxis().y, getToolAxis().z,
                currentPoint.x, currentPoint.y, currentPoint.z,
                getToolOrigin(currentPoint).x,                      getToolOrigin(currentPoint).y,
getToolOrigin(currentPoint).z
            );
        }

        public Point3D changeCoordinates(Point3D currentPoint, double step) {
            double[] xyz = MathUtils.multiply3X3with3(
                    new double[][]{
                            {1, 0                          , 0                },
                            {0, Math.cos(getAlpha()) , Math.sin(getAlpha())   },
                            {0, -Math.sin(getAlpha()), Math.cos(getAlpha())   },
                    },
                    new double[] {
                            currentPoint.x, currentPoint.y, currentPoint.z
                    }
            );

            return new Point3D(xyz[0], xyz[1], xyz[2]);
        }

        public Point2D.Double computeSecondOrderProfile(Point3D currentPointInToolSystem) {
            double h = currentPointInToolSystem.z;
            double r = Math.sqrt(
                    currentPointInToolSystem.x * currentPointInToolSystem.x +
                    currentPointInToolSystem.y * currentPointInToolSystem.y
            );
            return new Point2D.Double(r, h);
        }
    }
```

Codul sursă complet – aplicație scula cremalieră

```
package ro.ugal.profiles.approximation.datamodel.polynomial;

import org.jscience.mathematics.number.Float64;
import org.jscience.mathematics.vector.DenseVector;
import org.jscience.mathematics.vector.Float64Matrix;
import org.jscience.mathematics.vector.Float64Vector;
import org.jscience.mathematics.vector.Matrix;
import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;
import ro.ugal.profiles.approximation.ui.Toolbar;

import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.geom.GeneralPath;
import java.awt.geom.Point2D;
import java.text.DecimalFormat;
import java.util.HashMap;
import java.util.Map;

public class PolynomialCurve extends TheoreticalProfile {
    private float EPSILON = 0.001f;

    private int degree = 1;

    private double[] coeficientsY;
    private double[] coeficientsX;

    private Toolbar toolbar;

    private double profileLenght = 0;

    public PolynomialCurve(int degree, Toolbar toolbar) {
        this.degree = degree;
        this.toolbar = toolbar;
    }


    /**
     * Find out polynom coeficients by evaluating the polynom for different values for lambda,
     * and creating the equation system:
     * sum(i=n,0) Ai (lambda^n) * ((1 - lambda)^(n-1));
     * @param lambda
     * @param values
     * @param x if true coeficients are for X curve; if false - coeficients for Y curve
     */
    private void computeCoeficients(double[] lambda, double[] values,boolean x) {
        double[][] matrix = new double[degree + 1][degree + 1];
        for (int i = 0; i <= degree; i++) {
            for (int j = 0; j <= degree; j++) {
                matrix[i][j] = Math.pow(lambda[i], j) * Math.pow(1 - lambda[i], degree - j);
            }
        }
        Float64Matrix M = Float64Matrix.valueOf(matrix);
        Matrix             result             =             M.solve(Float64Matrix.valueOf(new
Float64Vector[]{Float64Vector.valueOf(values)}).transpose());

        if (x) {
            //result for coeficientsX
            DenseVector vector = (DenseVector) result.getColumn(0);
            coeficientsX = new double[degree + 1];
            for (int i = 0; i < degree + 1; i++) {
                coeficientsX[i] = ((Float64) vector.get(i)).doubleValue();
            }
        } else {
            //result for coeficientsY
            DenseVector vector = (DenseVector) result.getColumn(0);
            coeficientsY = new double[degree + 1];
            for (int i = 0; i < degree + 1; i++) {
                coeficientsY[i] = ((Float64) vector.get(i)).doubleValue();
            }
        }
    }

    /**
     * Find out polynom coeficients by evaluating the polynom for different values for lambda,
     * and creating the equation system:
```

```
 * sum(i=n,0) Ai (lambda^n) * ((1 - lambda)^(n-1));
 * @param lambda
 * @param values
 */
public void computeCoeficientsX(double[] lambda, double[] values) {
    computeCoeficients(lambda, values, true);
}

/**
 * Find out polynom coeficients by evaluating the polynom for different values for lambda,
 * and creating the equation system:
 * sum(i=n,0) Ai (lambda^n) * ((1 - lambda)^(n-1));
 * @param lambda
 * @param values
 */
public void computeCoeficientsY(double[] lambda, double[] values) {
    computeCoeficients(lambda, values, false);
}

TheoreticalProfile theoreticalProfile;

double[] lambdaCurrent = null;

static double x3 = 0;
static double y3 = 0;

public void computeProfileLenght(TheoreticalProfile theoreticalProfile, double radius) {
    Object[] result1 = theoreticalProfile.getPointAtRelativePosition(0);
    Object[] result2 = theoreticalProfile.getPointAtRelativePosition(1);

    double x1 = ((Point2D.Double) result1[0]).x;
    double y1 = ((Point2D.Double) result1[0]).y;

    double x2 = ((Point2D.Double) result2[0]).x;
    double y2 = ((Point2D.Double) result2[0]).y;

    profileLenght = radius * Math.sqrt(
                (x1 - x2) * (x1 - x2) +
                (y1 - y2) * (y1 - y2)
            ) / TheoreticalProfile.PROFILE_RADIOUS;
}

public void interpolate(TheoreticalProfile theoreticalProfile, double[] lambda) {

    lambdaCurrent = lambda;
    this.theoreticalProfile = theoreticalProfile;
    double[] valuesX = new double[this.degree + 1];
    double[] valuesY = new double[this.degree + 1];
    for (int i = 0; i < lambda.length; i++) {
        Object[] result = theoreticalProfile.getPointAtRelativePosition(lambda[i]);

        Point2D.Double point1 = (Point2D.Double) result[0];
        Point2D.Double point2 = (Point2D.Double) result[1];
        Point2D.Double anotherPoint = TheoreticalProfile.
                enwrappingCondition(point1, point2);

        valuesX[i] = anotherPoint.getX();
        valuesY[i] = anotherPoint.getY();
    }
    coeficientsY = new double[3];
    coeficientsX = new double[3];

    computeCoeficients(lambda, valuesX, true);
    computeCoeficients(lambda, valuesY, false);
}

public void paint(Graphics2D g) {
    //approximated profile (by poles)
    GeneralPath polinomial = new GeneralPath();
    Point2D.Double polinomialFirst = null;
    Point2D.Double polinomialLast = null;

    double errorMax = 0;
    double PAS = 20;

    double xMin = 10000;
    double xMax = -10000;
```

```
for (float lambda = 0; lambda <= 1; lambda += EPSILON) {
    double x = 0;
    double y = 0;
    for (int j = 0; j <= degree; j++) {
        x += coeficientsX[j] * Math.pow(lambda, j) * Math.pow(1 - lambda, degree - j);
        y += coeficientsY[j] * Math.pow(lambda, j) * Math.pow(1 - lambda, degree - j);
    }
    if (lambda == 0) {
        polinomialFirst = new Point2D.Double(x, y);
        polinomial.moveTo(x, y);
    } else {
        polinomialLast = new Point2D.Double(x, y);
        polinomial.lineTo(x, y);
    }


}
g.setColor(Color.BLACK);

//theorethical profile translated form piece to tool
profilePointsList.clear();
GeneralPath theoretical = new GeneralPath();
Point2D.Double theoreticalFirst = null;
Point2D.Double theoreticalLast = null;
int index = 0;
for (double lambda = 0; lambda < 1-EPSILON; lambda += EPSILON) {
    Object[] result = theoreticalProfile.getPointAtRelativePosition(lambda);

    Point2D.Double point1 = (Point2D.Double) result[0];
    Point2D.Double point2 = (Point2D.Double) result[1];

    Point2D.Double anotherPoint = TheoreticalProfile.
            enwrappingCondition(point1, point2);

    double x = anotherPoint.getX();
    double y = anotherPoint.getY();
    profilePointsList.add(new Point2D.Double(x, y));
    if (lambda == 0) {
        theoreticalFirst = new Point2D.Double(x, y);
        theoretical.moveTo(x, y);
    } else {
        theoreticalLast = new Point2D.Double(x, y);
        theoretical.lineTo(x, y);
    }
    if (x < xMin) {
        xMin = x;
    }
    if (x > xMax) {
        xMax = x;
    }
    index++;
}

//finaly draw profiles (over everything else)
widthX = xMax - xMin;
minX = xMin;
maxX = xMax;

g.setColor(Color.ORANGE.darker());
g.draw(polinomial);
{
    g.fill(
            new Ellipse2D.Double(
                    polinomialFirst.getX() - 2d/zoomValue,
                    polinomialFirst.getY() - 2d/zoomValue,
                    4d/zoomValue , 4d/zoomValue
            )
    );
}
{
    g.fill(
            new Ellipse2D.Double(
                    polinomialLast.getX() - 2d/zoomValue,
                    polinomialLast.getY() - 2d/zoomValue,
                    4d/zoomValue , 4d/zoomValue
            )
```

```
        );
    }

    g.setColor(Color.GREEN.darker());
    g.draw(theoretical);
    {
        g.fill(
                new Ellipse2D.Double(
                        theoreticalFirst.getX() - POINT_MARKER/zoomValue,
                        theoreticalFirst.getY() - POINT_MARKER/zoomValue,
                        2*POINT_MARKER/zoomValue , 2*POINT_MARKER/zoomValue
                )
        );
    }
    {
        g.fill(
                new Ellipse2D.Double(
                        theoreticalLast.getX() - 2*2d/zoomValue,
                        theoreticalLast.getY() - 2*2d/zoomValue,
                        2*POINT_MARKER/zoomValue , 2*POINT_MARKER/zoomValue
                )
        );
    }

}

public double widthX;

public double minX;

public double maxX;

public static String report = "";


public void calculate(int frequency) {
    report +="<table border='1' width='50%'>" +
                "<tr>" +
                    "<td style='' align='right'><b>Lambda</b></td>" +

                    "<td style='' align='right'><b>Approx(xi)</b></td>" +
                    "<td style='' align='right'><b>Approx(ita)</b></td>" +

                    "<td style='' align='right'><b>Tool profile(xi)</b></td>" +
                    "<td style='' align='right'><b>Tool profile(ita)</b></td>" +

                    "<td style='' align='right'><b>Err.</b></td>" +
                    "<td style='' align='right'><b>Phi</b></td>" +
                "</tr>";

    //approximated profile (by poles)

    double errorMax = 0;
    double PAS = 20;


    int indexAfisare = 0;
    for (double lambdau = 0; lambdau <= 1000; lambdau++) {
        double lambda = lambdau / 1000;
        Object[] result = theoreticalProfile.getCorrectProfile() != null ?
                theoreticalProfile.getCorrectProfile().getPointAtRelativePosition(lambda):
                theoreticalProfile.getPointAtRelativePosition(lambda);

        Point2D.Double point1 = (Point2D.Double) result[0];
        Point2D.Double point2 = (Point2D.Double) result[1];

        Point2D.Double anotherPoint = TheoreticalProfile.
                enwrappingCondition(point1, point2);

        double phi = TheoreticalProfile.getAnglePhi(point1, point2);

        double x_Profile = anotherPoint.getX();
        double y_Profile = anotherPoint.getY();

        double distClosest = 1000;
        double xClosest = 1000;
        double yClosest = 1000;
```

```
                double lambdaClosest = 1000;
                for (int r = -500; r <= 500; r+=1) {
                    double lambda_Approximation = lambda + ((double) r) * EPSILON;

                    if (lambda_Approximation < 0) {
                        continue;
                    }
                    if (lambda_Approximation > 1) {
                        continue;
                    }

                    double x_Poly = 0;
                    double y_Poly= 0;
                    for (int j = 0; j <= degree; j++) {
                        x_Poly += coeficientsX[j] * Math.pow(lambda_Approximation, j) * Math.pow(1
- lambda_Approximation, degree - j);
                        y_Poly += coeficientsY[j] * Math.pow(lambda_Approximation, j) * Math.pow(1
- lambda_Approximation, degree - j);
                    }

                    double dist = Math.sqrt (
                        (x_Profile - x_Poly) * (x_Profile - x_Poly) +
                        (y_Profile - y_Poly) * (y_Profile - y_Poly)
                    );

                    if (dist < distClosest) {
                        lambdaClosest = lambda_Approximation;
                        distClosest = dist;
                        xClosest = x_Poly;
                        yClosest = y_Poly;
                    }

                }

                DecimalFormat f = new DecimalFormat("0.0000");

                boolean displayBoldValues = false;

                if (degree == 2 && (lambdau == 0 || lambdau == 500 || lambdau == 1000)) {
                    displayBoldValues = true;
                }
                if (degree == 3 && (lambdau == 0 || lambdau == 333 || lambdau == 666 || lambdau ==
1000)) {
                    displayBoldValues = true;
                }
                if (degree == 4 && (lambdau == 0 || lambdau == 255 || lambdau == 500 || lambdau ==
750 || lambdau == 1000)) {
                    displayBoldValues = true;
                }

                if(indexAfisare % frequency == 0 || displayBoldValues ) {
                report += "<tr>" +
                        "<td>"           +(displayBoldValues?"<b>":"")           +           lambda
+(displayBoldValues?"</b>":"")+ "</td>" +

                        "<td nowrap='true'  style='' align='right'>"+(displayBoldValues?"<b>":"")+
f.format(toolbar.getRadius() + yClosest * theoreticalProfile.ratio) +(displayBoldValues?"</b>":"")+
"</td>" +
                        "<td nowrap='true' style='' align='right'>" +(displayBoldValues?"<b>":"")+
f.format(xClosest * theoreticalProfile.ratio) +(displayBoldValues?"</b>":"")+ "</td>" +

                        "<td nowrap='true' style='' align='right'>" +(displayBoldValues?"<b>":"")+
f.format(toolbar.getRadius() + y_Profile * theoreticalProfile.ratio) +(displayBoldValues?"</b>":"")+
"</td>" +
                        "<td nowrap='true' style='' align='right'>" +(displayBoldValues?"<b>":"")+
f.format(x_Profile * theoreticalProfile.ratio) +(displayBoldValues?"</b>":"")+ "</td>" +

                        "<td nowrap='true' style='' align='right'>" +(displayBoldValues?"<b>":"")+
f.format(distClosest * theoreticalProfile.ratio) +(displayBoldValues?"</b>":"")+ "</td>" +

                        "<td>"            +(displayBoldValues?"<b>":"")+            f.format(phi)
+(displayBoldValues?"</b>":"")+ "</td>" +

                        "</tr>";
                }
                indexAfisare ++;
```

```
                if (distClosest * theoreticalProfile.ratio > errorMax) {
                    errorMax = distClosest * theoreticalProfile.ratio;
                }
            }
            report += "</table>";


            DecimalFormat f = new DecimalFormat("0.000000");
            computeProfileLenght(theoreticalProfile,                                        ((Double)
toolbar.radiusSpinner.getValue()).doubleValue());
            toolbar.infos.append(
                    "Lambda:" + f.format(lambdaCurrent[1]) + " \n" +
                    "Max error:" + f.format(errorMax)  + " mm\n" +
                    "Profile lenght:" + f.format(profileLenght) + " mm\n" +
                    "Radius:" + toolbar.radiusSpinner.getValue() + "mm\n" +
                    "-----------------------------------------------------------------\n"
            );
            report += "<pre>" +
                    "Lambda:" + f.format(lambdaCurrent[1]) + " \n" +
                    "Max error:" + f.format(errorMax)  + " mm\n" +
                    "Profile lenght:" + f.format(profileLenght) + " mm\n" +
                    "Radius:" + toolbar.radiusSpinner.getValue() + "mm\n" +
                    "-----------------------------------------------------------------\n" +
                    "</pre>";

            Map pointAtRelativePositions = new HashMap();
            for (double lambdau = 0; lambdau <= 1000; lambdau++) {
                double lambda = lambdau / 1000;
                Object[] result = theoreticalProfile.getPointAtRelativePosition(lambda);
                pointAtRelativePositions.put(lambda, result);
            }
            long startTimeTheoretical = System.currentTimeMillis();
            for (int xxx = 0; xxx < 100; xxx++) {
                for (double lambdau = 0; lambdau <= 1000; lambdau++) {
                    double lambda = lambdau / 1000;
                    Object[] result = (Object[]) pointAtRelativePositions.get(lambda);

                    Point2D.Double point1 = (Point2D.Double) result[0];
                    Point2D.Double point2 = (Point2D.Double) result[1];

                    Point2D.Double anotherPoint = TheoreticalProfile.
                            enwrappingCondition(point1, point2);

                    double x_Profile = anotherPoint.getX();
                    double y_Profile = anotherPoint.getY();
                }
            }
            long finishTimeTheoretical = System.currentTimeMillis();


            Map lambdasCoef = new HashMap();
            for (double lambdau = 0; lambdau <= 1000; lambdau++) {
                double lambda = lambdau / 1000;
                Object[] result = theoreticalProfile.getPointAtRelativePosition(lambda);
                lambdasCoef.put(lambda, result);
            }

            long startTimeApprox = System.currentTimeMillis();
            for (int xxx = 0; xxx < 100; xxx++) {
                PolynomialCurve polCurve = new PolynomialCurve(
                                ((Double) toolbar.polyDegreeSpinner.getValue()).intValue(),
                                toolbar
                        );
                polCurve.interpolate(
                        theoreticalProfile,
                        toolbar.getLambdaValues()
                );
                for (double lambdau = 0; lambdau <= 1000; lambdau++) {
                    double lambda = lambdau / 1000;

                    double x_Poly = 0;
                    double y_Poly= 0;
                    for (int j = 0; j <= polCurve.degree; j++) {
                        double constant = pow(lambda, j) * pow(1 - lambda, polCurve.degree - j);
                        x_Poly += polCurve.coeficientsX[j] * constant;
                        y_Poly += polCurve.coeficientsY[j] * constant;
                    }
```

```
                }
            }
            long finishTimeApprox = System.currentTimeMillis();

            System.out.println("TIME COMPUTING THEORETICAL PROFILE:" + (finishTimeTheoretical -
startTimeTheoretical));
            System.out.println("TIME COMPUTING APPROX PROFILE:" + (finishTimeApprox -
startTimeApprox));

        }

        private double pow(double v, int power) {
            double ret = 1;
            for(int i = 0; i < power; i++) {
                ret = ret * v;
            }
            return ret;
        }
    }
```

```
    package ro.ugal.profiles.approximation.datamodel;

    /**
     * Mathemathical function approximation by Taylor polynoms.
     */
    public class MathTA {
        /**
         * Approximate SIN function using Taylor polynoms.
         * @param theta the angle
         * @param degree the degree of the approximating Taylor polynom;
         * if degree = 0 best approximation provided by Java Math implementation is used.
         * @return approximation of sin(theta)
         */
        public static double sin(double theta, int degree) {
            if (degree == 0) {
                return java.lang.Math.sin(theta);
            } else {
                double sum = 0;
                for (int n = 0; n < degree; n++) {
                    sum += java.lang.Math.pow(theta, 2 * n + 1) * java.lang.Math.pow(-1 ,n) /
fact(2 * n + 1);
                }
                return sum;
            }
        }

        /**
         * Approximate COS function using Taylor polynoms.
         * @param theta the angle
         * @param degree the degree of the approximating Taylor polynom;
         * if degree = 0 best approximation provided by Java Math implementation is used.
         * @return approximation of cos(theta)
         */
        public static double cos(double theta, int degree) {
            if (degree == 0) {
                return java.lang.Math.cos(theta);
            } else {
                double sum = 0;
                for (int n = 0; n < degree; n++) {
                    sum += java.lang.Math.pow(theta, 2 * n) * java.lang.Math.pow(-1 ,n) / fact(2 *
n);
                }
                return sum;
            }
        }

        /**
         * Approximate ATAN function using Taylor polynoms.
         * @param theta the angle
         * @param degree the degree of the approximating Taylor polynom;
         * if degree = 0 best approximation provided by Java Math implementation is used.
         * @return approximation of atan(theta)
         */
        public static double atan(double theta, int degree) {
            if (degree == 0) {
                return java.lang.Math.atan(theta);
```

```
            } else {
                double sum = 0;
                for (int n = 0; n < degree; n++) {
                    sum += java.lang.Math.pow(theta, 2 * n + 1) * java.lang.Math.pow(-1 ,n) / (2 *
n + 1);
                }
                return sum;
            }
        }

        private static double fact(int i) {
            if (i == 0) {
                return 1;
            } else {
                return i * fact(i - 1);
            }
        }
    }
```

```
    package ro.ugal.profiles.approximation.datamodel;


    import java.awt.*;
    import java.awt.geom.Ellipse2D;
    import java.awt.geom.Line2D;
    import java.awt.geom.Point2D;
    import java.io.Serializable;
    import java.util.ArrayList;
    import java.util.List;

    public class TheoreticalProfile implements Serializable {

        public static final double POINT_MARKER = 3;

        public static double PROFILE_RADIOUS = 300;

        public boolean showTheoreticalProfileTolerance = false;

        public boolean showControlPoints = false;

        public double lambda1 = 0;

        public double lambda2 = 1;

        public double radious = -1;

        protected int insertMidControlPointsCount = 0;

        public double ratio = 1;
        public double zoomValue = 0;

        TheoreticalProfile correctProfile;

        protected java.util.List profilePointsList = new ArrayList();

        protected java.util.List profileControlPoints = new ArrayList();


        public TheoreticalProfile() {

        }

        public java.awt.geom.Point2D.Double getProfile(int i) {
            return (java.awt.geom.Point2D.Double) profilePointsList.get(i);
        }

        public java.util.List getProfilePointsList() {
            return profilePointsList;
        }

        public void setProfilePointsList(List profilePointsList) {
            this.profilePointsList = profilePointsList;
        }

        public List getProfileControlPoints() {
            return profileControlPoints;
```

```
        }

        public void setProfileControlPoints(List profileControlPoints) {
            this.profileControlPoints = profileControlPoints;
        }

        public double getAngle(int index) {
            if (index == 0) {
                double x1 = getProfile(index).getX();
                double x2 = getProfile(index + 1).getX();

                double y1 = getProfile(index).getY();
                double y2 = getProfile(index + 1).getY();

                double dx = x1 - x2;
                double dy = y1 - y2;
                double a1 = Math.atan2(dy, dx) + Math.PI / 2;
                return a1;
            } else if (index == profilePointsList.size() - 1) {
                double x1 = getProfile(index - 1).getX();
                double x2 = getProfile(index).getX();

                double y1 = getProfile(index - 1).getY();
                double y2 = getProfile(index).getY();

                double dx = x1 - x2;
                double dy = y1 - y2;
                double a2 = Math.atan2(dy, dx) + Math.PI / 2;
                return a2;
            } else {
                double x1 = getProfile(index).getX();
                double x2 = getProfile(index + 1).getX();

                double y1 = getProfile(index).getY();
                double y2 = getProfile(index + 1).getY();

                double dx = x1 - x2;
                double dy = y1 - y2;
                double a1 = Math.atan2(dy, dx) + Math.PI / 2;

                x1 = getProfile(index - 1).getX();
                x2 = getProfile(index).getX();

                y1 = getProfile(index - 1).getY();
                y2 = getProfile(index).getY();

                dx = x1 - x2;
                dy = y1 - y2;
                double a2 = Math.atan2(dy, dx) + Math.PI / 2;
                return a2;
            }
        }

        public void paint(Graphics2D g2d, boolean onlyProfile) {
            if (!onlyProfile) {
                g2d.setColor(Color.BLUE.darker().darker());
            }
            for (int i = 0; i < profilePointsList.size() - 1; i++) {
                g2d.setColor(Color.BLUE.darker().darker());
                g2d.draw(
                        new Line2D.Double(
                                getProfile(i),
                                getProfile(i + 1)
                        )
                );

                g2d.setColor(new Color(128, 128, 255));
                //virtual lines
                g2d.draw(
                        new Line2D.Double(
                                getProfile(i).x  +  (getProfile(i  +  1).x  -  getProfile(i).x)  *
lambda1,
                                getProfile(i).y  +  (getProfile(i  +  1).y  -  getProfile(i).y)  *
lambda1,
                                getProfile(i).x,
                                getProfile(i).y
                        )
```

```
                );

                g2d.draw(
                        new Line2D.Double(
                                getProfile(i + 1).x,
                                getProfile(i + 1).y,
                                getProfile(i).x  +  (getProfile(i  +  1).x  -  getProfile(i).x)  *
lambda2,
                                getProfile(i).y  +  (getProfile(i  +  1).y  -  getProfile(i).y)  *
lambda2
                        )
                );
            }
            if (!onlyProfile) {
                if (showControlPoints) {
                    for (int i = 0; i < profilePointsList.size(); i++) {
                        g2d.draw(
                                new Ellipse2D.Double(
                                        getProfile(i).getX() - 2,
                                        getProfile(i).getY() - 2,
                                        4d, 4d
                                )
                        );
                    }
                }


                for (int i = 0; i < profileControlPoints.size(); i++) {
                    g2d.setColor(Color.RED);
                    Point2D.Double controlPoint = (Point2D.Double) profileControlPoints.get(i);
                    g2d.draw(
                            new Line2D.Double(
                                    controlPoint.getX() - 0.1,
                                    controlPoint.getY() - 0.1,
                                    controlPoint.getX() + 0.1,
                                    controlPoint.getY() + 0.1
                            )
                    );
                    g2d.draw(
                            new Line2D.Double(
                                    controlPoint.getX() + 0.1,
                                    controlPoint.getY() - 0.1,
                                    controlPoint.getX() - 0.1,
                                    controlPoint.getY() + 0.1
                            )
                    );
                }
                g2d.setColor(Color.BLUE.darker().darker());
                {
                    int i = 0;
                    g2d.fill(
                            new Ellipse2D.Double(
                                    getProfile(i).getX() - POINT_MARKER / zoomValue,
                                    getProfile(i).getY() - POINT_MARKER / zoomValue,
                                    2 * POINT_MARKER / zoomValue, 2 * POINT_MARKER / zoomValue
                            )
                    );
                }
                {
                    int i = profilePointsList.size() - 1;
                    g2d.fill(
                            new Ellipse2D.Double(
                                    getProfile(i).getX() - POINT_MARKER / zoomValue,
                                    getProfile(i).getY() - POINT_MARKER / zoomValue,
                                    2 * POINT_MARKER / zoomValue, 2 * POINT_MARKER / zoomValue
                            )
                    );
                }

            }
        }


        public void addPoint(double x, double y) {
            //invalidate existing theoretical profile for the tool each time a new control points
is added
            insertMidControlPointsCount = 0;
```

```
            profilePointsList.add(new java.awt.geom.Point2D.Double(x, y));
        }


        /**
         * Return the point X at a distance lambda * TotalCurveLength form the start point.
         *
         * @param lambda between 0 and 1
         * @return the determinated point X and tangent at the given point.
         */
        public Object[] getPointAtRelativePosition(double lambda) {
            double totalDistance = 0;
            double tangentAngle = 0;
            for (int i = 0; i < profilePointsList.size() - 1; i++) {
                Point2D.Double point1 = new Point2D.Double(
                        getProfile(i).x + (getProfile(i + 1).x - getProfile(i).x) * lambda1,
                        getProfile(i).y + (getProfile(i + 1).y - getProfile(i).y) * lambda1
                );
                Point2D.Double point2 = new Point2D.Double(
                        getProfile(i).x + (getProfile(i + 1).x - getProfile(i).x) * lambda2,
                        getProfile(i).y + (getProfile(i + 1).y - getProfile(i).y) * lambda2
                );
                totalDistance += point1.distance(point2);
            }

            double cumulativeDistance = 0;
            double lastDistance = 0;
            int position = 0;
            for (int i = 0; i < profilePointsList.size() - 1; i++) {

                cumulativeDistance += lastDistance;
                Point2D.Double point1 = new Point2D.Double(
                        getProfile(i).x + (getProfile(i + 1).x - getProfile(i).x) * lambda1,
                        getProfile(i).y + (getProfile(i + 1).y - getProfile(i).y) * lambda1
                );
                Point2D.Double point2 = new Point2D.Double(
                        getProfile(i).x + (getProfile(i + 1).x - getProfile(i).x) * lambda2,
                        getProfile(i).y + (getProfile(i + 1).y - getProfile(i).y) * lambda2
                );
                tangentAngle = Math.atan2(point2.y - point1.y, point2.x - point1.x);
                lastDistance = point1.distance(point2);

                position = i;
                if (cumulativeDistance + lastDistance > totalDistance * lambda) {
                    break;
                }
            }

            Point2D.Double point1 = new Point2D.Double(
                    getProfile(position).x + (getProfile(position + 1).x - getProfile(position).x)
* lambda1,
                    getProfile(position).y + (getProfile(position + 1).y - getProfile(position).y)
* lambda1
            );
            Point2D.Double point2 = new Point2D.Double(
                    getProfile(position).x + (getProfile(position + 1).x - getProfile(position).x)
* lambda2,
                    getProfile(position).y + (getProfile(position + 1).y - getProfile(position).y)
* lambda2
            );

            double x1 = point1.x;
            double x2 = point2.x;

            double y1 = point1.y;
            double y2 = point2.y;

            double k = (lambda * totalDistance - cumulativeDistance) / lastDistance;
            if (k > 0.5) {
                return new Object[]{
                        new Point2D.Double(x1 + k * (x2 - x1), y1 + k * (y2 - y1)),
                        new Point2D.Double(x1, y1)
                };
            } else {
                return new Object[]{
                        new Point2D.Double(x1 + k * (x2 - x1), y1 + k * (y2 - y1)),
                        new Point2D.Double(x2, y2)
```

```
                };
            }
        }


        public static double getAnglePhi(Point2D.Double point1, Point2D.Double point2) {

            double X1 = point1.x;
            double Y1 = point1.y;

            double alpha;
            double perpendicularSlope;
            double X2;
            double Y2;

            if ((point1.getX() - point2.getX() != 0) && (point1.getY() - point2.getY() != 0)) {
                alpha = ((point1.getY() - point2.getY()) / (point1.getX() - point2.getX()));
                perpendicularSlope = -1 / alpha;
                X2 = X1 + 1000;
                Y2 = perpendicularSlope * (X2 - X1) + Y1;
            } else if ((point1.getY() - point2.getY() != 0)) {
                perpendicularSlope  =  -(point1.getX()  -  point2.getX())  /  (point1.getY()  -
point2.getY());
                X2 = X1 + 1000;
                Y2 = perpendicularSlope * (X2 - X1) + Y1;
            } else {
                X2 = X1 + 1000;
                Y2 = Y1;
            }


            double dx = X2 - X1;
            double dy = Y2 - Y1;
            double dr = Math.sqrt(dx * dx + dy * dy);
            double D = X1 * Y2 - X2 * Y1;


            double  x1  =  (D  *  dy  +  Math.signum(dy)  *  dx  *  Math.sqrt(PROFILE_RADIOUS  *
PROFILE_RADIOUS * dr * dr - D * D)) / (dr * dr);
            double y1 = (-D * dx + Math.abs(dy) * Math.sqrt(PROFILE_RADIOUS * PROFILE_RADIOUS * dr
* dr - D * D)) / (dr * dr);

            double  x2  =  (D  *  dy  -  Math.signum(dy)  *  dx  *  Math.sqrt(PROFILE_RADIOUS  *
PROFILE_RADIOUS * dr * dr - D * D)) / (dr * dr);
            double y2 = (-D * dx - Math.abs(dy) * Math.sqrt(PROFILE_RADIOUS * PROFILE_RADIOUS * dr
* dr - D * D)) / (dr * dr);

            double phi = -MathTA.atan(x2 / y2, 0);

            return phi;
        }


        public static Point2D.Double enwrappingCondition(
                Point2D.Double point1,
                Point2D.Double point2) {

            double X1 = point1.x;
            double Y1 = point1.y;


            double alpha;
            double perpendicularSlope;
            double X2;
            double Y2;

            if ((point1.getX() - point2.getX() != 0) && (point1.getY() - point2.getY() != 0)) {
                alpha = ((point1.getY() - point2.getY()) / (point1.getX() - point2.getX()));
                perpendicularSlope = -1 / alpha;
                X2 = X1 + 1000;
                Y2 = perpendicularSlope * (X2 - X1) + Y1;
            } else if ((point1.getY() - point2.getY() != 0)) {
                perpendicularSlope  =  -(point1.getX()  -  point2.getX())  /  (point1.getY()  -
point2.getY());
                X2 = X1 + 1000;
                Y2 = perpendicularSlope * (X2 - X1) + Y1;
            } else {
```

```
                X2 = X1 + 1000;
                Y2 = Y1;
            }


            double dx = X2 - X1;
            double dy = Y2 - Y1;
            double dr = Math.sqrt(dx * dx + dy * dy);
            double D = X1 * Y2 - X2 * Y1;


            double x1 = (D * dy + Math.signum(dy) * dx * Math.sqrt(PROFILE_RADIOUS *
PROFILE_RADIOUS * dr * dr - D * D)) / (dr * dr);
            double y1 = (-D * dx + Math.abs(dy) * Math.sqrt(PROFILE_RADIOUS * PROFILE_RADIOUS * dr
* dr - D * D)) / (dr * dr);

            double x2 = (D * dy - Math.signum(dy) * dx * Math.sqrt(PROFILE_RADIOUS *
PROFILE_RADIOUS * dr * dr - D * D)) / (dr * dr);
            double y2 = (-D * dx - Math.abs(dy) * Math.sqrt(PROFILE_RADIOUS * PROFILE_RADIOUS * dr
* dr - D * D)) / (dr * dr);

            double phi = -MathTA.atan(x2 / y2, 0);
            double gama = MathTA.atan((x2 - point1.x) / (y2 - point1.y), 0);

            double xRot = point1.y * Math.sin(phi) + point1.x * Math.cos(phi) + phi *
PROFILE_RADIOUS;
            double yRot = (point1.y * Math.cos(phi) - point1.x * Math.sin(phi));

            return new Point2D.Double(xRot, yRot);
        }

        public String toString() {
            if (radious == -1) {
                //segment
                return lambda1 + "<<" +
                        "[" + getProfile(0).x * ratio + ", " + getProfile(0).y * ratio + "]" +
                        "-" +
                        "[" + getProfile(1).x * ratio + ", " + getProfile(1).y * ratio + "]" +
                        ">>" + lambda2;
            } else {
                //arc
                return lambda1 + "<<" +
                        "(" + getProfile(profilePointsList.size() - 1).x * ratio + ", " +
getProfile(profilePointsList.size() - 1).y * ratio + ")" +
                        "-<" + radious * ratio + ">-" +
                        "(" + getProfile(profilePointsList.size() - 1).x * ratio + ", " +
getProfile(profilePointsList.size() - 1).y * ratio + ")" +
                        ">>" + lambda2;
            }
        }

        public void resize(double radiusRatio) {
            ratio = radiusRatio;
        }

        public void setZoomLevel(double zoomValue) {
            this.zoomValue = zoomValue;
        }

        public TheoreticalProfile getCorrectProfile() {
            return correctProfile;
        }

        public void setCorrectProfile(TheoreticalProfile correctProfile) {
            this.correctProfile = correctProfile;
        }
    }
```

```
    package ro.ugal.profiles.approximation.datamodel;

    import java.awt.*;
    import java.awt.geom.Ellipse2D;
    import java.awt.geom.GeneralPath;
    import java.awt.geom.Line2D;
    import java.util.ArrayList;
```

```
public class ToolProfile {

    private double deltaGrande = 10;

    private double deltaMinor  = -10;

    public static double PROFILE_RADIOUS = 300;

    public boolean showToolProfileTolerance = false;
    public boolean showControlPoints = false;

    java.util.List profilePointsList = new ArrayList();

       java.awt.geom.Point2D.Double getProfile(int i) {
           return (java.awt.geom.Point2D.Double) profilePointsList.get(i);
       }

       public void paint(Graphics2D g2d) {
           g2d.setColor(Color.BLACK);

           g2d.draw(
                   new Line2D.Double(-300, 0, 300, 0)
           );
           g2d.draw(
                   new Line2D.Double(0, -300, 0, 300)
           );
           g2d.draw(
               new Ellipse2D.Double(
                   -300, -300, 600, 600
               )
           );

           g2d.setColor(Color.BLUE);
           for (int i = 0; i < profilePointsList.size() - 1; i++) {
               g2d.draw(
                   new Line2D.Double(
                       getProfile(i),
                       getProfile(i + 1)
                   )
               );
           }

           if (showControlPoints) {
               for (int i = 0; i < profilePointsList.size() ; i++) {
                   g2d.draw(
                       new Ellipse2D.Double(
                           getProfile(i).getX() - 2,
                           getProfile(i).getY() - 2,
                           4d , 4d
                       )
                   );
               }
           }
       }
}
```

```
package ro.ugal.profiles.approximation.ui;


public class Applet extends  javax.swing.JApplet {
    public void init() {
        this.getContentPane().add(new Main());
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Point2D;
```

```
public class DialogArc extends JDialog implements ActionListener {
    private JLabel x1Label = new JLabel("Y1");
    private JLabel y1Label = new JLabel("X1");
    private JLabel x2Label = new JLabel("Y2");
    private JLabel y2Label = new JLabel("X2");

    private JLabel radiusLabel = new JLabel("R");

    private JLabel lambda1Label = new JLabel("Lambda 1");
    private JLabel lambda2Label = new JLabel("Lambda 2");

    private JTextField x1 = new JTextField();
    private JTextField y1 = new JTextField();
    private JTextField x2 = new JTextField();
    private JTextField y2 = new JTextField();

    private JTextField radius = new JTextField();

    private JTextField lambda1 = new JTextField("0");
    private JTextField lambda2 = new JTextField("1");

    private JButton preview = new JButton("preview");
    private JButton cancel = new JButton("done");

    private int index = -1;

    Toolbar toolbar = null;

    public DialogArc(Toolbar toolbar, int index) {
        this.index = index;
        TheoreticalProfile tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);
        this.x1.setText("" + tp.getProfile(0).x *  toolbar.getRadiusRatio());
        this.y1.setText("" + tp.getProfile(0).y *  toolbar.getRadiusRatio());
        this.x2.setText(""   +   tp.getProfile(tp.getProfilePointsList().size()   -   1).x   *
toolbar.getRadiusRatio());
        this.y2.setText(""   +   tp.getProfile(tp.getProfilePointsList().size()   -   1).y   *
toolbar.getRadiusRatio());
        this.radius.setText("" + tp.radious *  toolbar.getRadiusRatio());
        this.lambda1.setText("" + tp.lambda1);
        this.lambda2.setText("" + tp.lambda2);

        x1Label.setBounds(10,  10, 20, 20);
             x1.setBounds(30,  10, 50, 20);
        y1Label.setBounds(80,  10, 20, 20);
             y1.setBounds(100, 10, 50, 20);

        x2Label.setBounds(10,  30, 20, 20);
             x2.setBounds(30,  30, 50, 20);
        y2Label.setBounds(80,  30, 20, 20);
             y2.setBounds(100, 30, 50, 20);

        radiusLabel.setBounds(10, 50, 20, 20);
             radius.setBounds(30, 50, 50, 20);

        lambda1Label.setBounds(10, 80, 100, 20);
        lambda1.setBounds(110, 80, 100, 20);

        lambda2Label.setBounds(10, 100, 100, 20);
        lambda2.setBounds(110, 100, 100, 20);

        preview.setBounds(110, 130, 80, 20);
         cancel.setBounds(210, 130, 80, 20);

        this.setLayout(null);
        this.setSize(300, 300);
        this.add(x1Label);
        this.add(y1Label);
        this.add(x2Label);
        this.add(y2Label);
        this.add(radiusLabel);
        this.add(lambda1Label);
        this.add(lambda2Label);

        this.add(preview);
        this.add(cancel);

        preview.addActionListener(this);
```

```
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(x2);
    this.add(y2);
    this.add(radius);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public DialogArc(Toolbar toolbar) {

    x1Label.setBounds(10,  10, 20, 20);
         x1.setBounds(30,  10, 50, 20);
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);

    x2Label.setBounds(10,  30, 20, 20);
         x2.setBounds(30,  30, 50, 20);
    y2Label.setBounds(80,  30, 20, 20);
         y2.setBounds(100, 30, 50, 20);

    radiusLabel.setBounds(10, 50, 20, 20);
         radius.setBounds(30, 50, 50, 20);

    lambda1Label.setBounds(10, 80, 100, 20);
    lambda1.setBounds(110, 80, 100, 20);

    lambda2Label.setBounds(10, 100, 100, 20);
    lambda2.setBounds(110, 100, 100, 20);

    preview.setBounds(110, 130, 80, 20);
     cancel.setBounds(210, 130, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(x2Label);
    this.add(y2Label);
    this.add(radiusLabel);
    this.add(lambda1Label);
    this.add(lambda2Label);

    this.add(preview);
    this.add(cancel);

    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(x2);
    this.add(y2);
    this.add(radius);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public boolean previewAdded = false;

public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == cancel) {
        this.dispose();
    } else if (actionEvent.getSource() == preview) {

        if (index != -1) {
            //edit
```

```
            } else {
                //add
                if (previewAdded) {
                    //first remove last theoretical profile
                    toolbar.theoreticalProfile.remove(
                            toolbar.theoreticalProfile.size() - 1
                    );
                }
            }
            //add this profile
            TheoreticalProfile tp = getProfile();

            tp.resize(toolbar.getRadiusRatio());
            tp.lambda1 = Double.parseDouble(lambda1.getText());
            tp.lambda2 = Double.parseDouble(lambda2.getText());
            if (index != -1) {
                toolbar.theoreticalProfile.set(index, tp);
            } else {
                toolbar.theoreticalProfile.add(tp);
            }
            toolbar.segments.setListData(
                toolbar.theoreticalProfile.toArray()
            );
            previewAdded = true;
        }
        toolbar.graphicCanvas.repaint();
    }

    private TheoreticalProfile getProfile() {
        TheoreticalProfile tp = new TheoreticalProfile();

        double rVal  = Double.parseDouble(radius.getText()) / toolbar.getRadiusRatio();

        double x1Val = Double.parseDouble(x1.getText()) / toolbar.getRadiusRatio();
        double y1Val = Double.parseDouble(y1.getText()) / toolbar.getRadiusRatio();

        double x2Val = Double.parseDouble(x2.getText()) / toolbar.getRadiusRatio();
        double y2Val = Double.parseDouble(y2.getText()) / toolbar.getRadiusRatio();

        double xMiddle = (x1Val + x2Val) / 2;
        double yMiddle = (y1Val + y2Val) / 2;

        double d = Math.sqrt(
                (xMiddle - x1Val) * (xMiddle - x1Val) +
                (yMiddle - y1Val) * (yMiddle - y1Val)
        );

        double e = Math.sqrt(
                rVal * rVal - d * d
        );
        if (rVal < 0) {
            e = -e;
            rVal = -rVal;
        }

        double gama = Math.atan( - (x2Val - x1Val) / (y2Val - y1Val));
        double xCenter = e * Math.cos(gama) + xMiddle;
        double yCenter = e * Math.sin(gama) + yMiddle;

        Point2D.Double pointCenter = new Point2D.Double(xCenter, yCenter);

        tp.radious = rVal;

        for (double lambda = 0; lambda <= 1; lambda += 0.005) {
            Point2D.Double currentPoint = new Point2D.Double(
                x1Val + lambda * (x2Val - x1Val),
                y1Val + lambda * (y2Val - y1Val)
            );

            double RADIOUS = pointCenter.distance(currentPoint);
            double x = (rVal) * (currentPoint.x - pointCenter.x) / RADIOUS + pointCenter.x;
            double y = (rVal) * (currentPoint.y - pointCenter.y) / RADIOUS + pointCenter.y;
            tp.addPoint(x, y);
        }
        return tp;
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class DialogEpicic extends JDialog implements ActionListener {
    private JLabel x1Label = new JLabel("Y1");
    private JLabel y1Label = new JLabel("X1");

    private JLabel radiusMinorLabel = new JLabel("r");
    private JLabel radiusMajorLabel = new JLabel("R");
    private JLabel alphaStartLabel = new JLabel("alpha1");
    private JLabel alphaStopLabel= new JLabel("alpha2");

    private JCheckBox directionBackwards = new JCheckBox("direction <");

    private JLabel lambda1Label = new JLabel("Lambda 1");
    private JLabel lambda2Label = new JLabel("Lambda 2");

    private JTextField x1 = new JTextField();
    private JTextField y1 = new JTextField();

    private JTextField radiusMinor = new JTextField();
    private JTextField radiusMajor = new JTextField();
    private JTextField alphaStart = new JTextField();
    private JTextField alphaStop = new JTextField();

    private JTextField lambda1 = new JTextField("0");
    private JTextField lambda2 = new JTextField("1");

    private JButton preview = new JButton("preview");
    private JButton cancel = new JButton("done");

    private int index = -1;

    Toolbar toolbar = null;

    public DialogEpicic(Toolbar toolbar, int index) {
        this.index = index;
        TheoreticalProfile tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);
        this.x1.setText("" + tp.getProfile(0).x *  toolbar.getRadiusRatio());
        this.y1.setText("" + tp.getProfile(0).y *  toolbar.getRadiusRatio());

        this.radiusMinor.setText("" + tp.radious *  toolbar.getRadiusRatio());
        this.lambda1.setText("" + tp.lambda1);
        this.lambda2.setText("" + tp.lambda2);

        x1Label.setBounds(10,  10, 20, 20);
             x1.setBounds(30,  10, 50, 20);
        y1Label.setBounds(80,  10, 20, 20);
             y1.setBounds(100, 10, 50, 20);

        radiusMinorLabel.setBounds(10,  30, 20, 20);
             radiusMinor.setBounds(30,  30, 50, 20);
        radiusMajorLabel.setBounds(80,  30, 20, 20);
             radiusMajor.setBounds(100, 30, 50, 20);

        alphaStartLabel.setBounds(10,  50, 50, 20);
             alphaStart.setBounds(60,  50, 50, 20);
        alphaStopLabel.setBounds(110,  50, 50, 20);
             alphaStop.setBounds(160,  50, 50, 20);

        directionBackwards.setBounds(10,  70, 200, 20);

        lambda1Label.setBounds(10, 100, 100, 20);
             lambda1.setBounds(110, 100, 100, 20);

        lambda2Label.setBounds(10, 120, 100, 20);
             lambda2.setBounds(110, 120, 100, 20);
        preview.setBounds(110, 150, 80, 20);
         cancel.setBounds(210, 150, 80, 20);
```

```
    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(radiusMinorLabel);
    this.add(radiusMajorLabel);
    this.add(alphaStartLabel);
    this.add(directionBackwards);
    this.add(lambda1Label);
    this.add(lambda2Label);

    this.add(preview);
    this.add(cancel);

    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(radiusMinor);
    this.add(radiusMajor);
    this.add(alphaStart);
    this.add(alphaStop);
    this.add(directionBackwards);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public DialogEpicic(Toolbar toolbar) {

    x1Label.setBounds(10,  10, 20, 20);
         x1.setBounds(30,  10, 50, 20);
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);

    radiusMinorLabel.setBounds(10,  30, 20, 20);
         radiusMinor.setBounds(30,  30, 50, 20);
    radiusMajorLabel.setBounds(80,  30, 20, 20);
         radiusMajor.setBounds(100, 30, 50, 20);

    alphaStartLabel.setBounds(10,  50, 50, 20);
         alphaStart.setBounds(60,  50, 50, 20);
    alphaStopLabel.setBounds(110,  50, 50, 20);
         alphaStop.setBounds(160,  50, 50, 20);

    directionBackwards.setBounds(10,  70, 200, 20);

    lambda1Label.setBounds(10, 100, 100, 20);
    lambda1.setBounds(110, 100, 100, 20);

    lambda2Label.setBounds(10, 120, 100, 20);
    lambda2.setBounds(110, 120, 100, 20);

    preview.setBounds(110, 150, 80, 20);
     cancel.setBounds(210, 150, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(radiusMinorLabel);
    this.add(radiusMajorLabel);
    this.add(alphaStartLabel);
    this.add(directionBackwards);
    this.add(lambda1Label);
    this.add(lambda2Label);

    this.add(preview);
    this.add(cancel);

    preview.addActionListener(this);
    cancel.addActionListener(this);
```

```
    this.add(x1);
    this.add(y1);
    this.add(radiusMinor);
    this.add(radiusMajor);
    this.add(alphaStart);
    this.add(alphaStop);
    this.add(directionBackwards);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public boolean previewAdded = false;

public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == cancel) {
        this.dispose();
    } else if (actionEvent.getSource() == preview) {

        if (index != -1) {
            //edit
        } else {
            //add
            if (previewAdded) {
                //first remove last theoretical profile
                toolbar.theoreticalProfile.remove(
                        toolbar.theoreticalProfile.size() - 1
                );
            }
        }
        //add this profile
        TheoreticalProfile tp = getProfile();

        tp.resize(toolbar.getRadiusRatio());
        tp.lambda1 = Double.parseDouble(lambda1.getText());
        tp.lambda2 = Double.parseDouble(lambda2.getText());
        if (index != -1) {
            toolbar.theoreticalProfile.set(index, tp);
        } else {
            toolbar.theoreticalProfile.add(tp);
        }
        toolbar.segments.setListData(
            toolbar.theoreticalProfile.toArray()
        );
        previewAdded = true;
    }
    toolbar.graphicCanvas.repaint();
}

private TheoreticalProfile getProfile() {
    TheoreticalProfile tp = new TheoreticalProfile();



    double rMinor = Double.parseDouble(radiusMinor.getText()) / toolbar.getRadiusRatio();
    double rMajor = Double.parseDouble(radiusMajor.getText()) / toolbar.getRadiusRatio();

    double x1Val = Double.parseDouble(x1.getText()) / toolbar.getRadiusRatio();
    double y1Val = Double.parseDouble(y1.getText()) / toolbar.getRadiusRatio();

    double alphaStartValue = Double.parseDouble(alphaStart.getText());
    double alphaStopValue  = Double.parseDouble(alphaStop.getText());

    double k = rMajor / rMinor;

    tp.radious = rMinor;
    double phi = alphaStartValue;
    int index = 0;

    double deltaX = 0;
    double deltaY = 0;

    double sign = 1;
    if (directionBackwards.isSelected()) {
```

```
                sign = -1;
            }

            while (phi < alphaStopValue) {
                index ++;
                phi += Math.PI / 300;
                double xx = rMinor * (k + 1) * ( Math.cos(phi) - Math.cos((k + 1)*phi) / ( k +
1));
                double yy = rMinor * (k + 1) * ( Math.sin(phi) - Math.sin((k + 1)*phi) / ( k +
1));

                double x = sign * (+xx * Math.cos(Math.PI / 2) + yy * Math.sin(Math.PI / 2)) ;
                double y = (-xx * Math.sin(Math.PI / 2) + yy * Math.cos(Math.PI / 2)) ;

                double r = Math.sqrt(x * x + y * y);

                if (tp.getProfilePointsList().size() == 0) {
                    //first point, compute deltaX, deltaY
                    deltaX = x - x1Val;
                    deltaY = y - y1Val;
                }
                tp.addPoint(x - deltaX, y - deltaY);

            }
            return tp;
        }
    }
```

```
    package ro.ugal.profiles.approximation.ui;

    import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

    import javax.swing.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;

    public class DialogEvolvent extends JDialog implements ActionListener {
        private JLabel x1Label = new JLabel("Y1");
        private JLabel y1Label = new JLabel("X1");

        private JLabel radiusMinorLabel = new JLabel("Ri");
        private JLabel radiusMajorLabel = new JLabel("Re");
        private JLabel radiusBazaLabel = new JLabel("Rb");
        private JCheckBox directionBackwards = new JCheckBox("direction <");

        private JLabel lambda1Label = new JLabel("Lambda 1");
        private JLabel lambda2Label = new JLabel("Lambda 2");

        private JTextField x1 = new JTextField();
        private JTextField y1 = new JTextField();

        private JTextField radiusMinor = new JTextField();
        private JTextField radiusMajor = new JTextField();
        private JTextField radiusBaza = new JTextField();

        private JTextField lambda1 = new JTextField("0");
        private JTextField lambda2 = new JTextField("1");

        private JButton preview = new JButton("preview");
        private JButton cancel = new JButton("done");

        private int index = -1;

        Toolbar toolbar = null;

        public DialogEvolvent(Toolbar toolbar, int index) {
            this.index = index;
            TheoreticalProfile tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);
            this.x1.setText("" + tp.getProfile(0).x *  toolbar.getRadiusRatio());
            this.y1.setText("" + tp.getProfile(0).y *  toolbar.getRadiusRatio());
            this.radiusMinor.setText("" + tp.radious *  toolbar.getRadiusRatio());
            this.lambda1.setText("" + tp.lambda1);
            this.lambda2.setText("" + tp.lambda2);

            x1Label.setBounds(10,  10, 20, 20);
                 x1.setBounds(30,  10, 50, 20);
```

```
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);


    radiusBazaLabel.setBounds(10,  30, 20, 20);
         radiusBaza.setBounds(30,  30, 50, 20);

    radiusMinorLabel.setBounds(80,  30, 20, 20);
         radiusMinor.setBounds(100,  30, 50, 20);

    radiusMajorLabel.setBounds(150, 30, 20, 20);
         radiusMajor.setBounds(170, 30, 50, 20);

    directionBackwards.setBounds(10, 50, 200, 20);

    lambda1Label.setBounds(10, 80, 100, 20);
    lambda1.setBounds(110, 80, 100, 20);

    lambda2Label.setBounds(10, 100, 100, 20);
    lambda2.setBounds(110, 100, 100, 20);

    preview.setBounds(110, 130, 80, 20);
     cancel.setBounds(210, 130, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(radiusMinorLabel);
    this.add(radiusMajorLabel);
    this.add(radiusBazaLabel);
    this.add(directionBackwards);
    this.add(lambda1Label);
    this.add(lambda2Label);

    this.add(preview);
    this.add(cancel);

    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(radiusMinor);
    this.add(radiusMajor);
    this.add(radiusBaza);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public DialogEvolvent(Toolbar toolbar) {

    x1Label.setBounds(10,  10, 20, 20);
         x1.setBounds(30,  10, 50, 20);
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);


    radiusBazaLabel.setBounds(10,  30, 20, 20);
         radiusBaza.setBounds(30,  30, 50, 20);

    radiusMinorLabel.setBounds(80,  30, 20, 20);
         radiusMinor.setBounds(100,  30, 50, 20);

    radiusMajorLabel.setBounds(150,  30, 20, 20);
         radiusMajor.setBounds(170, 30, 50, 20);

    directionBackwards.setBounds(10, 50, 200, 20);

    lambda1Label.setBounds(10, 80, 100, 20);
    lambda1.setBounds(110, 80, 100, 20);

    lambda2Label.setBounds(10, 100, 100, 20);
```

```
    lambda2.setBounds(110, 100, 100, 20);

    preview.setBounds(110, 130, 80, 20);
     cancel.setBounds(210, 130, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(radiusMinorLabel);
    this.add(radiusMajorLabel);
    this.add(radiusBazaLabel);
    this.add(directionBackwards);
    this.add(lambda1Label);
    this.add(lambda2Label);

    this.add(preview);
    this.add(cancel);

    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(radiusMinor);
    this.add(radiusMajor);
    this.add(radiusBaza);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public boolean previewAdded = false;

public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == cancel) {
        this.dispose();
    } else if (actionEvent.getSource() == preview) {

        if (index != -1) {
            //edit
        } else {
            //add
            if (previewAdded) {
                //first remove last theoretical profile
                toolbar.theoreticalProfile.remove(
                        toolbar.theoreticalProfile.size() - 1
                );
            }
        }
        //add this profile
        TheoreticalProfile tp = getProfile();

        tp.resize(toolbar.getRadiusRatio());
        tp.lambda1 = Double.parseDouble(lambda1.getText());
        tp.lambda2 = Double.parseDouble(lambda2.getText());
        if (index != -1) {
            toolbar.theoreticalProfile.set(index, tp);
        } else {
            toolbar.theoreticalProfile.add(tp);
        }
        toolbar.segments.setListData(
            toolbar.theoreticalProfile.toArray()
        );
        previewAdded = true;
    }
    toolbar.graphicCanvas.repaint();
}

private TheoreticalProfile getProfile() {
    TheoreticalProfile tp = new TheoreticalProfile();

    double x1Val = Double.parseDouble(x1.getText()) / toolbar.getRadiusRatio();
    double y1Val = Double.parseDouble(y1.getText()) / toolbar.getRadiusRatio();
```

```
        double deltaX = 0;
        double deltaY = 0;

        double rMinor = Double.parseDouble(radiusMinor.getText()) / toolbar.getRadiusRatio();
        double rBaza  = Double.parseDouble(radiusBaza.getText()) / toolbar.getRadiusRatio();
        double rMajor = Double.parseDouble(radiusMajor.getText()) / toolbar.getRadiusRatio();


        tp.radious = rMinor;
        double phi = -Math.PI/4;
        int index = 0;
        double sign = 1;
        if (directionBackwards.isSelected()) {
            sign = -1;
        }

        while (true) {
            index ++;
            phi += Math.PI / 1000;
            double xx = (rBaza * Math.cos(phi) + rBaza * Math.abs(phi) * Math.sin(phi));
            double yy = (rBaza * Math.sin(phi) - rBaza * phi * Math.cos(phi));


            double x = sign * (+xx * Math.cos(Math.PI / 2) + yy * Math.sin(Math.PI / 2)) ;
            double y = (-xx * Math.sin(Math.PI / 2) + yy * Math.cos(Math.PI / 2)) ;

            double r = Math.sqrt(x * x + y * y);

            if (r < rMinor) {
                continue;
            }

            if (r > rMajor) {
                break;
            }


            if (tp.getProfilePointsList().size() == 0) {
                //first point, compute deltaX, deltaY
                deltaX = x - x1Val;
                deltaY = y - y1Val;
            }
            tp.addPoint(x - deltaX, y - deltaY);
        }
        return tp;
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Point2D;
import java.text.DecimalFormat;

public class DialogInspection extends JDialog implements ActionListener, ChangeListener {

    JLabel   measuredPointsLabel          = new JLabel("Measured points");
    public JSpinner measuredPointsSpinner = new JSpinner(
        new SpinnerNumberModel(2d, 1d, 50d, 1d)
    );

    private JLabel[] lambdaLabel    = new JLabel[3];

    private JTextField[] lambda     = new JTextField[3];

    private JTextArea result        = new JTextArea();

    private JButton ok = new JButton("OK");
```

```
private int index = -1;

Toolbar toolbar = null;

TheoreticalProfile tp;
public DialogInspection(Toolbar toolbar, int index) {
    this.index = index;
    tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);

    measuredPointsLabel.setBounds(0, 0,200,20 );
    measuredPointsSpinner.setBounds(200,0,100,20);
    this.add(measuredPointsLabel);
    this.add(measuredPointsSpinner);

    measuredPointsSpinner.addChangeListener(this);


    this.setLayout(null);
    this.setSize(400, 300);

    createElements();

    result.setBounds(80, 20, 180, 100);
    this.add(result);

    this.add(ok);
    ok.setBounds(210, 130, 80, 20);
    ok.addActionListener(this);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);


    this.setModal(true);
    this.toolbar = toolbar;
    createElements();
    this.setVisible(true);
}

private void createElements() {
    for (int i = 0; i < lambdaLabel.length; i++) {
        if (lambda[i] != null) {
            this.remove(lambda[i]);
        }
        if (lambdaLabel[i] != null) {
            this.remove(lambdaLabel[i]);
        }
    }
    int degree = ((Double) measuredPointsSpinner.getValue()).intValue();

    lambdaLabel = new JLabel[(int) degree + 1];

    lambda = new JTextField[(int) degree + 1];

    for (int i = 0; i < degree; i++) {
        lambdaLabel[i] = new JLabel("L"+i);
        lambda[i] = new JTextField();
        lambdaLabel[i].setBounds(10,  20 + i*20, 20, 20);
             lambda[i].setBounds(30,  20 + i*20, 50, 20);
        this.add(lambdaLabel[i]);
        this.add(lambda[i]);
    }

    this.validate();
    this.repaint();
    this.invalidate();
}


public boolean previewAdded = false;

public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == ok) {
        String resultStr = "";
        for (int i = 0; i < ((Double) measuredPointsSpinner.getValue()).intValue(); i++) {
            double l = Double.parseDouble(lambda[i].getText());
```

```
                Point2D.Double point = (Point2D.Double) tp.getPointAtRelativePosition(l)[0];
                DecimalFormat f = new DecimalFormat("0.0000");
                resultStr += f.format(point.getX()) + "; " + f.format(point.getY()) + "\n";
            }
            result.setText(resultStr);
        }
    }

    public void stateChanged(ChangeEvent changeEvent) {
        if (changeEvent.getSource() == measuredPointsSpinner) {
            createElements();
        }
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import org.jscience.mathematics.number.Float64;
import org.jscience.mathematics.vector.DenseVector;
import org.jscience.mathematics.vector.Float64Matrix;
import org.jscience.mathematics.vector.Float64Vector;
import org.jscience.mathematics.vector.Matrix;
import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Point2D;

public class DialogPoly extends JDialog implements ActionListener, ChangeListener {

    JLabel   polyDegreeLabel          = new JLabel("Polynom degree");
    public JSpinner polyDegreeSpinner = new JSpinner(
        new SpinnerNumberModel(2d, 1d, 50d, 1d)
    );

    JLabel   measuredPointsLabel          = new JLabel("Measured points");
    public JSpinner measuredPointsSpinner = new JSpinner(
        new SpinnerNumberModel(2d, 1d, 50d, 1d)
    );

    private JLabel[] xLabel = new JLabel[3];
    private JLabel[] yLabel = new JLabel[3];

    private JTextField[] x = new JTextField[3];
    private JTextField[] y = new JTextField[3];

    private JButton preview = new JButton("preview");
    private JButton cancel = new JButton("done");

    private int index = -1;

    Toolbar toolbar = null;

    public DialogPoly(Toolbar toolbar, int index) {
        this.index = index;
        TheoreticalProfile tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);

        this.setModal(true);
        this.toolbar = toolbar;
        this.setVisible(true);
    }

    private void createElements() {
        for (int i = 0; i < xLabel.length; i++) {
            if (xLabel[i] != null) {
                this.remove(xLabel[i]);
            }
            if (x[i] != null) {
                this.remove(x[i]);
            }
            if (yLabel[i] != null) {
                this.remove(yLabel[i]);
            }
```

```
            if (y[i] != null) {
                this.remove(y[i]);
            }
        }
        int degree = ((Double) measuredPointsSpinner.getValue()).intValue();

        xLabel = new JLabel[(int) degree];
        yLabel = new JLabel[(int) degree];

        x = new JTextField[(int) degree];
        y = new JTextField[(int) degree];

        for (int i = 0; i < degree; i++) {
            System.out.println(">>>:"+i);
            xLabel[i] = new JLabel("Y"+i);
            yLabel[i] = new JLabel("X"+i);
            x[i] = new JTextField();
            y[i] = new JTextField();
            xLabel[i].setBounds(10,  20 + i*20, 20, 20);
                 x[i].setBounds(30,  20 + i*20, 50, 20);
            yLabel[i].setBounds(80,  20 + i*20, 20, 20);
                 y[i].setBounds(100, 20 + i*20, 50, 20);
            this.add(xLabel[i]);
            this.add(x[i]);
            this.add(yLabel[i]);
            this.add(y[i]);
        }
        this.validate();
        this.repaint();
        this.invalidate();
    }

    public DialogPoly(Toolbar toolbar) {
        polyDegreeLabel.setBounds(0, 0,100,20 );
        polyDegreeSpinner.setBounds(100,0,100,20);
        this.add(polyDegreeLabel);
        this.add(polyDegreeSpinner);

        measuredPointsSpinner.setBounds(0, 0,200,20 );
        measuredPointsSpinner.setBounds(200,0,100,20);
        this.add(measuredPointsLabel);
        this.add(measuredPointsSpinner);

        polyDegreeSpinner.addChangeListener(this);
        measuredPointsSpinner.addChangeListener(this);

        createElements();

        preview.setBounds(210, 130, 80, 20);
        cancel.setBounds(310, 130, 80, 20);

        this.setLayout(null);
        this.setSize(400, 300);

        this.add(preview);
        this.add(cancel);

        //add.addActionListener(this);
        preview.addActionListener(this);
        cancel.addActionListener(this);

        this.setModal(true);
        this.toolbar = toolbar;
        this.setVisible(true);
    }

    public boolean previewAdded = false;

    public void actionPerformed(ActionEvent actionEvent) {

        if (actionEvent.getSource() == cancel) {
            this.dispose();
        } else if (actionEvent.getSource() == preview) {

            if (index != -1) {
                //edit
            } else {
```

```
            //add
            if (previewAdded) {
                //first remove last theoretical profile
                toolbar.theoreticalProfile.remove(
                        toolbar.theoreticalProfile.size() - 1
                );
            }
        }
        //add this profile
        TheoreticalProfile tp = getProfile();

        tp.resize(toolbar.getRadiusRatio());

        x[1].setText("" + (Double.parseDouble(x[1].getText()) + 0.1));
        y[1].setText("" + (Double.parseDouble(y[1].getText()) + 0.1));
        TheoreticalProfile gresit = getProfile();
        gresit.setCorrectProfile(tp);
        gresit.resize(toolbar.getRadiusRatio());
        if (index != -1) {
            toolbar.theoreticalProfile.set(index, gresit);
        } else {
            toolbar.theoreticalProfile.add(gresit);
        }
        toolbar.segments.setListData(
            toolbar.theoreticalProfile.toArray()
        );
        previewAdded = true;
    }
    toolbar.graphicCanvas.repaint();
}

private TheoreticalProfile getProfile() {
    TheoreticalProfile tp = new TheoreticalProfile();
    int degree = ((Double) polyDegreeSpinner.getValue()).intValue();
    int points = ((Double) measuredPointsSpinner.getValue()).intValue();

    for (int i = 0; i < points; i++) {
        Point2D.Double p1 = new Point2D.Double(
            Double.parseDouble(x[i].getText())/ toolbar.getRadiusRatio(),
            Double.parseDouble(y[i].getText())/ toolbar.getRadiusRatio()
        );
        tp.getProfileControlPoints().add(p1);
    }


    double totalDist = 0;
    for (int i = 1; i < points; i++) {
        Point2D.Double p1 = new Point2D.Double(
            Double.parseDouble(x[i-1].getText())/ toolbar.getRadiusRatio(),
            Double.parseDouble(y[i-1].getText())/ toolbar.getRadiusRatio()
        );
        Point2D.Double p2 = new Point2D.Double(
            Double.parseDouble(x[i].getText())/ toolbar.getRadiusRatio(),
            Double.parseDouble(y[i].getText())/ toolbar.getRadiusRatio()
        );
        totalDist += p1.distance(p2);
    }

    double[] lambda = new double[(int) degree + 1];

    lambda[0] = 0;
    lambda[degree] = 1;
    double cumultativeDist = 0;

    int index = 1;
    int indices[] = new int[degree + 1];
    indices[0] = 0;
    indices[degree] = points - 1;
    for (int i = 1; i < points - 1; i++) {
        Point2D.Double p1 = new Point2D.Double(
            Double.parseDouble(x[i-1].getText()) / toolbar.getRadiusRatio(),
            Double.parseDouble(y[i-1].getText()) / toolbar.getRadiusRatio()
        );
        Point2D.Double p2 = new Point2D.Double(
            Double.parseDouble(x[i].getText()) / toolbar.getRadiusRatio(),
            Double.parseDouble(y[i].getText()) / toolbar.getRadiusRatio()
        );
```

```
                Point2D.Double p3 = new Point2D.Double(
                    Double.parseDouble(x[i + 1].getText()) / toolbar.getRadiusRatio(),
                    Double.parseDouble(y[i + 1].getText()) / toolbar.getRadiusRatio()
                );

                cumultativeDist += p1.distance(p2);

                double fraction = totalDist * index / degree;

                if (    (cumultativeDist  <= fraction) &&
                        (cumultativeDist + p2.distance(p3) >= fraction)) {

                    if   (Math.abs(cumultativeDist  -  fraction)  <=  Math.abs(cumultativeDist  +
p2.distance(p3) - fraction)) {
                        lambda[index] = cumultativeDist / totalDist;
                        indices[index] = i;
                        index++;
                    } else {
                        lambda[index] = (cumultativeDist + p2.distance(p3)) / totalDist;
                        indices[index] = (i + 1);
                        index++;
                    }
                }
                if (cumultativeDist  >= fraction) {
                    lambda[index] = cumultativeDist / totalDist;
                    indices[index] = i;
                    index++;
                }
            }

            for (int i = 0; i <= degree; i++) {
                System.out.println("LAMBDA[" + i + "]: >" + lambda[i]);
                System.out.println("INDICES[" + i + "]: >" + indices[i]);
            }

            double[][] matrix = new double[(int) degree + 1][(int) degree + 1];
            for (int i = 0; i <= degree; i++) {
                for (int j = 0; j <= degree; j++) {
                    matrix[i][j] = Math.pow(lambda[i], j) * Math.pow(1 - lambda[i], degree - j);
                }
            }
            double[]  coeficientsY = new double[(int)degree + 1];
            double[]  coeficientsX = new double[(int)degree + 1];

            {
                double[] values = new double[(int) degree + 1];
                for (int i = 0; i <= degree; i++) {
                    values[i]        =        Double.parseDouble(x[indices[i]].getText())        /
toolbar.getRadiusRatio();
                }
                Float64Matrix M = Float64Matrix.valueOf(matrix);
                Matrix result = M.solve(
                        Float64Matrix.valueOf(
                                new Float64Vector[]{
                                        Float64Vector.valueOf(values)
                                }
                        ).transpose()
                );

                //result for coeficientsX
                DenseVector vector1 = (DenseVector) result.getColumn(0);

                for (int i = 0; i < degree + 1; i++) {
                    coeficientsX[i] = ((Float64) vector1.get(i)).doubleValue();
                }
            }

            {
                double[] values = new double[(int) degree + 1];
                for (int i = 0; i <= degree; i++) {
                    values[i]        =        Double.parseDouble(y[indices[i]].getText())        /
toolbar.getRadiusRatio();
                }
                Float64Matrix M = Float64Matrix.valueOf(matrix);
                Matrix result = M.solve(
                        Float64Matrix.valueOf(
                                new Float64Vector[]{
```

```
                                    Float64Vector.valueOf(values)
                                }
                        ).transpose()
                );

                //result for coeficientsX
                DenseVector vector1 = (DenseVector) result.getColumn(0);

                for (int i = 0; i < degree + 1; i++) {
                    coeficientsY[i] = ((Float64) vector1.get(i)).doubleValue();
                }
            }

            for (int i = 0; i < degree + 1; i++) {
                System.out.println("COEFICEINTS_X[" + i + "]=" + coeficientsX[i]);
            }

            for (int i = 0; i < degree + 1; i++) {
                System.out.println("COEFICEINTS_Y[" + i + "]=" + coeficientsY[i]);
            }


            double[] maxError = new double[points + 1];
            for (int i = 0; i < points; i++) {
                maxError[i] = 10000;
            }


            for (double l = 0; l <= 1; l += 0.005) {
                double xx = 0;
                double yy = 0;
                for (int j = 0; j <= degree; j++) {
                    xx += coeficientsX[j] * Math.pow(l, j) * Math.pow(1 - l, degree - j);
                    yy += coeficientsY[j] * Math.pow(l, j) * Math.pow(1 - l, degree - j);
                }
                for (int i = 0; i < points; i++) {
                    double dist = Math.sqrt(
                            (xx - Double.parseDouble(x[i].getText()) / toolbar.getRadiusRatio()) *
(xx - Double.parseDouble(x[i].getText()) / toolbar.getRadiusRatio())
                            +
                            (yy - Double.parseDouble(y[i].getText()) / toolbar.getRadiusRatio()) *
(yy - Double.parseDouble(y[i].getText()) / toolbar.getRadiusRatio())
                    );
                    if (dist < maxError[i]) {
                        maxError[i] = dist;
                    }
                }
                tp.addPoint(xx, yy);
            }

            for (int i = 0; i < points; i++) {
                System.out.println("ERROR(" + i + "): >" + maxError[i]);
            }

            return tp;
        }

        public void stateChanged(ChangeEvent changeEvent) {
            if (changeEvent.getSource() == measuredPointsSpinner) {
                createElements();
            }
        }
    }
```

```
    package ro.ugal.profiles.approximation.ui;

    import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;

    import javax.swing.*;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;

    public class DialogSegment extends JDialog implements ActionListener {

        private JLabel x1Label = new JLabel("Y1");
        private JLabel y1Label = new JLabel("X1");
```

```
private JLabel x2Label = new JLabel("Y2");
private JLabel y2Label = new JLabel("X2");

private JLabel lambda1Label = new JLabel("Lambda 1");
private JLabel lambda2Label = new JLabel("Lambda 2");

private JTextField x1 = new JTextField();
private JTextField y1 = new JTextField();
private JTextField x2 = new JTextField();
private JTextField y2 = new JTextField();

private JTextField lambda1 = new JTextField("0");
private JTextField lambda2 = new JTextField("1");



//private JButton add = new JButton("save");
private JButton preview = new JButton("preview");
private JButton cancel = new JButton("done");

private int index = -1;

Toolbar toolbar = null;

public DialogSegment(Toolbar toolbar, int index) {
    this.index = index;
    TheoreticalProfile tp = (TheoreticalProfile) toolbar.theoreticalProfile.get(index);
    this.x1.setText("" + tp.getProfile(0).x *  toolbar.getRadiusRatio());
    this.y1.setText("" + tp.getProfile(0).y *  toolbar.getRadiusRatio());
    this.x2.setText("" + tp.getProfile(1).x *  toolbar.getRadiusRatio());
    this.y2.setText("" + tp.getProfile(1).y *  toolbar.getRadiusRatio());
    this.lambda1.setText("" + tp.lambda1);
    this.lambda2.setText("" + tp.lambda2);

    x1Label.setBounds(10,  10, 20, 20);
         x1.setBounds(30,  10, 50, 20);
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);

    x2Label.setBounds(10,  30, 20, 20);
         x2.setBounds(30,  30, 50, 20);
    y2Label.setBounds(80,  30, 20, 20);
         y2.setBounds(100, 30, 50, 20);

    lambda1Label.setBounds(10, 80, 100, 20);
    lambda1.setBounds(110, 80, 100, 20);

    lambda2Label.setBounds(10, 100, 100, 20);
    lambda2.setBounds(110, 100, 100, 20);

        //add.setBounds(10,  130, 80, 20);
    preview.setBounds(110, 130, 80, 20);
     cancel.setBounds(210, 130, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(x2Label);
    this.add(y2Label);
    this.add(lambda1Label);
    this.add(lambda2Label);

    //this.add(add);
    this.add(preview);
    this.add(cancel);

    //add.addActionListener(this);
    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(x2);
    this.add(y2);
    this.add(lambda1);
    this.add(lambda2);
```

```
    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public DialogSegment(Toolbar toolbar) {

    x1Label.setBounds(10,  10, 20, 20);
         x1.setBounds(30,  10, 50, 20);
    y1Label.setBounds(80,  10, 20, 20);
         y1.setBounds(100, 10, 50, 20);

    x2Label.setBounds(10,  30, 20, 20);
         x2.setBounds(30,  30, 50, 20);
    y2Label.setBounds(80,  30, 20, 20);
         y2.setBounds(100, 30, 50, 20);

    lambda1Label.setBounds(10, 80, 100, 20);
    lambda1.setBounds(110, 80, 100, 20);

    lambda2Label.setBounds(10, 100, 100, 20);
    lambda2.setBounds(110, 100, 100, 20);

        //add.setBounds(10,  130, 80, 20);
    preview.setBounds(110, 130, 80, 20);
     cancel.setBounds(210, 130, 80, 20);

    this.setLayout(null);
    this.setSize(300, 300);
    this.add(x1Label);
    this.add(y1Label);
    this.add(x2Label);
    this.add(y2Label);
    this.add(lambda1Label);
    this.add(lambda2Label);

    //this.add(add);
    this.add(preview);
    this.add(cancel);

    //add.addActionListener(this);
    preview.addActionListener(this);
    cancel.addActionListener(this);

    this.add(x1);
    this.add(y1);
    this.add(x2);
    this.add(y2);
    this.add(lambda1);
    this.add(lambda2);

    this.setModal(true);
    this.toolbar = toolbar;
    this.setVisible(true);
}

public boolean previewAdded = false;

public void actionPerformed(ActionEvent actionEvent) {
    if (actionEvent.getSource() == cancel) {
        this.dispose();
    } else if (actionEvent.getSource() == preview) {

        if (index != -1) {
            //edit
        } else {
            //add
            if (previewAdded) {
                //first remove last theoretical profile
                toolbar.theoreticalProfile.remove(
                        toolbar.theoreticalProfile.size() - 1
                );
            }
        }
        //add this profile
        TheoreticalProfile tp = new TheoreticalProfile();
```

```
            tp.addPoint(
                Double.parseDouble(x1.getText()) / toolbar.getRadiusRatio(),
                Double.parseDouble(y1.getText()) / toolbar.getRadiusRatio()
            );
            tp.addPoint(
                Double.parseDouble(x2.getText()) / toolbar.getRadiusRatio(),
                Double.parseDouble(y2.getText()) / toolbar.getRadiusRatio()
            );
            tp.resize(toolbar.getRadiusRatio());
            tp.lambda1 = Double.parseDouble(lambda1.getText());
            tp.lambda2 = Double.parseDouble(lambda2.getText());
            if (index != -1) {
                toolbar.theoreticalProfile.set(index, tp);
            } else {
                toolbar.theoreticalProfile.add(tp);
            }
            toolbar.segments.setListData(
                toolbar.theoreticalProfile.toArray()
            );
            previewAdded = true;
        }
        toolbar.graphicCanvas.repaint();
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;
import ro.ugal.profiles.approximation.datamodel.polynomial.PolynomialCurve;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.print.PrinterException;

public class DisplayTableValuesDialog extends JDialog implements ActionListener {

    JEditorPane editor          = new JEditorPane();
    JButton     calculateErrors = new JButton("<html>Calc.<br> err.");
    JButton     print           = new JButton("Print");
    JSpinner    valuesFrequency = new JSpinner(
            new SpinnerNumberModel(50d, 1d, 1000d, 1d)
    );
    JLabel      valuesFrequencyLabel = new JLabel("Val. frequency");

    Toolbar toolbar;

    public Dimension getPreferredSize() {
        return new Dimension(600, 500);
    }

    public DisplayTableValuesDialog(Toolbar toolbar) {
        this.toolbar = toolbar;
        this.setLayout(null);
        this.setSize(600, 500);

        editor.setEditable(false);
        editor.setContentType("text/html");      // must specify HTML text
        editor.setText("");
        JScrollPane editorScroll = new JScrollPane(editor);
        editorScroll.setBounds(10, 10, 580, 400);

        this.add(editorScroll);

        valuesFrequencyLabel.setBounds(10, 420, 90, 20);
        this.add(valuesFrequencyLabel);

        valuesFrequency.setBounds(100, 420, 50, 20);
        this.add(valuesFrequency);

        this.calculateErrors.setBounds(150, 420, 100, 50);
        this.add(calculateErrors);
        calculateErrors.addActionListener(this);

        this.print.setBounds(250, 420, 100, 50);
```

```
            this.add(print);
            print.addActionListener(this);
        }

        public void actionPerformed(ActionEvent actionEvent) {
            if (actionEvent.getSource() == calculateErrors) {
                //do error calculation
                PolynomialCurve.report = "";
                for (int tp = 0; tp < toolbar.getTheoreticalProfile().size(); tp++) {
                    TheoreticalProfile       theoreticalProfile       =       (TheoreticalProfile)
toolbar.getTheoreticalProfile().get(tp);

                    PolynomialCurve polCurve = null;
                    if(theoreticalProfile.getProfilePointsList().size() > 1) {
                        polCurve = new PolynomialCurve(
                                ((Double)        toolbar.polyDegreeSpinner.getValue()).intValue(),
toolbar
                        );

                        polCurve.interpolate(
                                theoreticalProfile,
                                toolbar.getLambdaValues()
                        );
                        polCurve.calculate(
                                ((Double) this.valuesFrequency.getValue()).intValue()
                        );
                    }
                }
                editor.setText(
                        "<font size=\"10pt\">"+
                        PolynomialCurve.report +
                        "</font>"
                );
            }
            if (actionEvent.getSource() == print) {
                try {
                    this.editor.print();
                } catch (PrinterException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

```
    package ro.ugal.profiles.approximation.ui;

    import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;
    import ro.ugal.profiles.approximation.datamodel.polynomial.PolynomialCurve;

    import javax.swing.*;
    import java.awt.*;
    import java.awt.event.*;
    import java.awt.geom.AffineTransform;
    import java.awt.geom.Ellipse2D;
    import java.awt.geom.Line2D;
    import java.awt.geom.Point2D;
    import java.text.DecimalFormat;
    import java.util.ArrayList;

    public class GraphicCanvas extends JPanel
            implements MouseMotionListener, MouseListener, AdjustmentListener {

        Toolbar toolbar;
        public JScrollPane scrollcanvasContainer;
        private double mousePressedX;
        private double mousePressedY;

        private double scrollX;
        private double scrollY;

        public Dimension getPreferredSize() {
            return this.getSize();
        }

        public GraphicCanvas(Toolbar toolbar) {
            this.toolbar = toolbar;
```

```
            this.setSize(2000, 2000);
            this.addMouseListener(this);
            this.addMouseMotionListener(this);
            gr = (Graphics2D) this.getGraphics();
        }

        private void drawArraw(Graphics2D g,float x1,float y1,float x2,float y2) {
                java.awt.geom.Line2D.Float line1 = new java.awt.geom.Line2D.Float(x1,y1,x2,y2);
                g.draw(line1);

                java.awt.geom.GeneralPath arrow = new java.awt.geom.GeneralPath();
                arrow.moveTo(x2,y2);
                arrow.lineTo(x2-5,y2-10);
                arrow.curveTo(x2-5,y2-10,x2,y2-5,x2+5,y2-10);
                arrow.lineTo(x2,y2);
                arrow.closePath();
                g.fill(arrow);
            }

        private void drawArrawInverse(Graphics2D g,float x1,float y1,float x2,float y2) {
                java.awt.geom.Line2D.Float line1 = new
java.awt.geom.Line2D.Float(x1,y1,x1,(y1+y2)/2);
                g.draw(line1);

                java.awt.geom.Line2D.Float line2 = new
java.awt.geom.Line2D.Float(x1,(y1+y2)/2,x2,(y1+y2)/2);
                g.draw(line2);

                java.awt.geom.Line2D.Float line3 = new
java.awt.geom.Line2D.Float(x2,(y1+y2)/2,x2,y2);
                g.draw(line3);

                java.awt.geom.GeneralPath arrow = new java.awt.geom.GeneralPath();
                arrow.moveTo(x2,y2);
                arrow.lineTo(x2-10,y2-5);
                arrow.curveTo(x2-10,y2-5,x2-5,y2,x2-10,y2+5);
                arrow.lineTo(x2,y2);
                arrow.closePath();
                g.fill(arrow);
            }

        public void paintComponent(Graphics g) {
            this.setSize(
                    (int) (2000 * toolbar.getZoomValue()),
                    (int) (2000 * toolbar.getZoomValue()));



            Graphics2D g2d = (Graphics2D) g;
            g2d.setStroke(
                    new java.awt.BasicStroke((float) (1 / toolbar.getZoomValue()))
            );

            gr = g2d;
            g2d.setColor(Color.WHITE);
            g2d.fillRect(
                    0, 0,
                    this.getWidth(),
                    this.getHeight());


            g2d.setRenderingHint(
                    RenderingHints.KEY_ALPHA_INTERPOLATION,
                    RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);

            g2d.setRenderingHint(
                    RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);

            int scrollX = scrollcanvasContainer.getHorizontalScrollBar().getValue();
            int scrollY = scrollcanvasContainer.getVerticalScrollBar().getValue();

            //simetric scale transformation
            AffineTransform transfomation = AffineTransform.getTranslateInstance(
                       500 * toolbar.getZoomValue() - scrollX,
                       500 * toolbar.getZoomValue() - scrollY);
            transfomation.concatenate(
```

```
    AffineTransform.getScaleInstance(
                toolbar.getZoomValue(), toolbar.getZoomValue())
);

double Rp = 300;

g2d.setTransform(transfomation);

//draw grid
for (int i = -300; i < 300; i += 50) {
    if (i == 0) {
        continue;
    }

    g2d.setColor(Color.LIGHT_GRAY);
    g2d.draw(
            new Line2D.Double(-300, i, 300, i)
    );
    g2d.draw(
            new Line2D.Double(i, -300, i, 300)
    );

    g2d.setColor(Color.DARK_GRAY);
    String value = new DecimalFormat("#.##").format(i * toolbar.getRadiusRatio());
    g2d.setFont(g2d.getFont().deriveFont(AffineTransform.getScaleInstance(
                1/toolbar.getZoomValue(), 1/toolbar.getZoomValue())));
    g2d.drawString(value, -300, i);
    g2d.drawString(value, 300, i);

    g2d.drawString(value, i, 300);
    g2d.drawString(value, i, -300);
}

g2d.setColor(Color.BLACK);

g2d.draw(
        new Line2D.Double(-300, 0, 300, 0)
);
g2d.draw(
        new Line2D.Double(0, -300, 0, 300)
);
g2d.draw(
        new Ellipse2D.Double(
                -300, -300, 600, 600
        )
);

g2d.draw(
        new Line2D.Double(-300, 0, 300, 0)
);
g2d.draw(
        new Line2D.Double(0, -300, 0, 300)
);
g2d.draw(
        new Ellipse2D.Double(
                -300, -300, 600, 600
        )
);


g2d.setColor(new Color(128, 0, 0));
drawArraw(g2d, 0, -300, 0, -200);
g2d.drawString("\u03BE",10, -200);

drawArraw(g2d, 0, 0, 0, 100);
g2d.drawString("X,x",10, 100);

drawArrawInverse(g2d, 0, -300, 100, -300);
g2d.drawString("\u03B7",100, -310);

drawArrawInverse(g2d, 0, 0, 100, 0);
g2d.drawString("Y,y",100, 10);

g2d.setColor(Color.BLUE.darker());
AffineTransform clearTransformation = g2d.getTransform();
double totalWidth = 0;
double minX = 1000;
```

```
                double maxX = -1000;
                for (int u = 0; u < coordinatesMarker.size(); u++) {
                    Point2D.Double point = (Point2D.Double) coordinatesMarker.get(u);
                    g.drawString(
                            "("+point.getX()+","+point.getY()+")",
                            (int) point.getX(), (int) point.getY()
                    );
                }

                for (int tp = 0; tp < toolbar.getTheoreticalProfile().size(); tp++) {
                    TheoreticalProfile        theoreticalProfile        =        (TheoreticalProfile)
toolbar.getTheoreticalProfile().get(tp);
                    theoreticalProfile.setZoomLevel(toolbar.getZoomValue());
                    theoreticalProfile.paint(g2d, false);

                    g2d.setColor(Color.RED.darker());
                    PolynomialCurve polCurve = null;
                    if(theoreticalProfile.getProfilePointsList().size() > 1) {
                        polCurve = new PolynomialCurve(
                                ((Double) toolbar.polyDegreeSpinner.getValue()).intValue(), toolbar
                        );

                        polCurve.interpolate(
                                theoreticalProfile,
                                toolbar.getLambdaValues()
                        );
                        polCurve.setZoomLevel(toolbar.getZoomValue());
                        polCurve.paint(g2d);
                        if (polCurve.minX < minX) {
                            minX = polCurve.minX;
                        }
                        if (polCurve.maxX > maxX) {
                            maxX = polCurve.maxX;
                        }
                    }

                    double phiIncrement = Math.PI/100;

                    if (toolbar.showMoves.isSelected()) {
                        for (int i = -100; i <= 100; i++) {
                            AffineTransform at = new AffineTransform(transfomation);
                            at.concatenate(
                                    AffineTransform.getTranslateInstance(- Rp * phiIncrement * i, 0)
                            );

                            at.concatenate(
                                AffineTransform.getRotateInstance(phiIncrement * i)
                            );

                            g2d.setTransform(at);

                            g2d.setColor(new Color(0, 0, 255, 64));
                            theoreticalProfile.paint(g2d, true);
                        }
                    }

                    g2d.setColor(Color.LIGHT_GRAY);
                    if (pointCenter != null) {
                        g2d.fill(
                                new Ellipse2D.Double(
                                        pointCenter.x - 2, pointCenter.y - 2, 4 , 4
                                )
                        );
                    }
                    if (pointRadious != null) {
                        g2d.fill(
                                new Ellipse2D.Double(
                                        pointRadious.x - 2, pointRadious.y - 2, 4 , 4
                                )
                        );
                    }
                }


                totalWidth += maxX - minX;

                g2d.setTransform(clearTransformation);
```

```
}

public void setScrollPane(JScrollPane scrollcanvasContainer) {
    this.scrollcanvasContainer = scrollcanvasContainer;
    scrollcanvasContainer.getVerticalScrollBar().addAdjustmentListener(this);
    scrollcanvasContainer.getHorizontalScrollBar().addAdjustmentListener(this);
}

private Point2D.Double pointCenter  = null;
private Point2D.Double pointRadious = null;

public void mouseClicked(MouseEvent e) {
    if (e.getClickCount() == 2) {
        coordinatesMarker.clear();
        this.repaint();
    }
}

public void mouseEntered(MouseEvent e) {
    //nothing
}

public void mouseExited(MouseEvent e) {
    //nothing
}

public boolean drag = false;
public void mousePressed(MouseEvent e) {
    drag = false;
    if (e.getButton() == MouseEvent.BUTTON3) {
        addCoordinatesMarker(e.getX(), e.getY());
        this.repaint();
        return;
    }
    if (toolbar.zoomIn.isSelected()) {
        double deltaX = scrollcanvasContainer.getWidth() / 2 -
                (e.getX() - scrollcanvasContainer.getHorizontalScrollBar().getValue()) ;
        double deltaY = scrollcanvasContainer.getHeight() / 2 -
                (e.getY() - scrollcanvasContainer.getVerticalScrollBar().getValue());
        double deltaZoom = 50;
        isModifying = true;
        scrollcanvasContainer.getHorizontalScrollBar().setValue(
            (int) (scrollcanvasContainer.getHorizontalScrollBar().getValue() - deltaX)
        );
        scrollcanvasContainer.getVerticalScrollBar().setValue(
            (int) (scrollcanvasContainer.getVerticalScrollBar().getValue() - deltaY)
        );
        isModifying = false;
        toolbar.zoomSpinner.setValue(
            ((Double) toolbar.zoomSpinner.getValue()) + deltaZoom
        );
    } else if (toolbar.zoomOut.isSelected()) {
        double deltaX = scrollcanvasContainer.getWidth() / 2 -
                (e.getX() - scrollcanvasContainer.getHorizontalScrollBar().getValue()) ;
        double deltaY = scrollcanvasContainer.getHeight() / 2 -
                (e.getY() - scrollcanvasContainer.getVerticalScrollBar().getValue());
        scrollcanvasContainer.getHorizontalScrollBar().setValue(
            (int) (scrollcanvasContainer.getHorizontalScrollBar().getValue() - deltaX)
        );
        scrollcanvasContainer.getVerticalScrollBar().setValue(
            (int) (scrollcanvasContainer.getVerticalScrollBar().getValue() - deltaY)
        );
        double deltaZoom = 50;
        toolbar.zoomSpinner.setValue(
            ((Double) toolbar.zoomSpinner.getValue()) - deltaZoom
        );
    } else {
        mousePressedX = e.getX();
        mousePressedY = e.getY();
        scrollX = scrollcanvasContainer.getHorizontalScrollBar().getValue();
        scrollY = scrollcanvasContainer.getVerticalScrollBar().getValue();
        drag = true;
    }
    e.consume();
}

java.util.List coordinatesMarker = new ArrayList();
```

```
    private void addCoordinatesMarker(int x, int y) {
        int scrollX = scrollcanvasContainer.getHorizontalScrollBar().getValue();
        int scrollY = scrollcanvasContainer.getVerticalScrollBar().getValue();

        double zoom = toolbar.getZoomValue();

        coordinatesMarker.add(
                new Point2D.Double(
                        ((double) x) / zoom - 500,
                        ((double) y) / zoom - 500
                )
        );
    }

    public void mouseReleased(MouseEvent e) {
        if (drag) {
            double mouseDragX = e.getX();
            double mouseDragY = e.getY();

            scrollcanvasContainer.getHorizontalScrollBar().setValue(
                (int) (scrollX + (mousePressedX - mouseDragX))
            );
            scrollcanvasContainer.getVerticalScrollBar().setValue(
                (int) (scrollY + (mousePressedY - mouseDragY))
            );
            this.repaint();
            this.setCursor(POINTER);
        }
    }


    static Cursor HAND = new Cursor(Cursor.HAND_CURSOR);
    static Cursor POINTER = new Cursor(Cursor.DEFAULT_CURSOR);

    public void mouseDragged(MouseEvent e) {
    this.setCursor(HAND);
    double mouseDragX = e.getX();
    double mouseDragY = e.getY();
    isModifying = true;
    scrollcanvasContainer.getHorizontalScrollBar().setValue(
        (int) (scrollX + (mousePressedX - mouseDragX))
    );
    scrollcanvasContainer.getVerticalScrollBar().setValue(
        (int) (scrollY + (mousePressedY - mouseDragY))
    );
    isModifying = false;
    //this.repaint();
}

    public void mouseMoved(MouseEvent e) {
        //To change body of implemented methods use File | Settings | File Templates.
    }

    private static Graphics2D gr;

    public static Graphics2D getGraphics2d() {
        return gr;
    }

    public void resetCircleParam() {
        pointCenter = null;
        pointRadious = null;
    }

    boolean isModifying = false;

    public void adjustmentValueChanged(AdjustmentEvent adjustmentEvent) {
        if (isModifying) {
            return;
        }
        this.repaint();
    }


}
```

```
package ro.ugal.profiles.approximation.ui;

import javax.swing.*;

public class Main extends JPanel {

    GraphicCanvas graphicCanvas;
    private int mousePressedX;
    private int mousePressedY;
    private int scrollX;
    private int scrollY;

    public Main() {
        this.setSize(1000, 600);
        Toolbar toolbar = new Toolbar();

        graphicCanvas = new GraphicCanvas(toolbar);
        toolbar.setGraphicCanvas(graphicCanvas);

        JScrollPane scrollcanvasContainer = new JScrollPane();
        graphicCanvas.setScrollPane(scrollcanvasContainer);
        scrollcanvasContainer.setViewportView(graphicCanvas);
        scrollcanvasContainer.setBounds(0, 0, 1000, 500);

        toolbar.setBounds(0, 500, 1000, 200);

        this.setLayout(null);
        this.add(toolbar);
        this.add(scrollcanvasContainer);
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import ro.ugal.profiles.approximation.datamodel.TheoreticalProfile;
import ro.ugal.profiles.approximation.datamodel.polynomial.PolynomialCurve;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class Toolbar extends JPanel
        implements ChangeListener, ActionListener {

    /**GUI Components */
    JButton  clearProfileButton  = new JButton("Clear");
    JLabel   zoomLabel          = new JLabel("Zoom");
    JSpinner zoomSpinner        = new JSpinner(
        new SpinnerNumberModel(100d, 0d, 500000d, 1d)
    );

    JLabel   ployDegreeLabel          = new JLabel("Polynom degree");
    public JSpinner polyDegreeSpinner        = new JSpinner(
        new SpinnerNumberModel(2d, 1d, 50d, 1d)
    );

    JLabel   radiusLabel          = new JLabel("Radius");
    public JSpinner radiusSpinner        = new JSpinner(
        new SpinnerNumberModel(300d, 10d, 1000d, 1d)
    );

    JCheckBox showMoves              = new JCheckBox("Show moves");

    JComboBox addProfile             = new JComboBox();
    {
        addProfile.addItem("--select profile to add--");
        addProfile.addItem("Add segment");
        addProfile.addItem("Add arc");
```

```
    addProfile.addItem("Add evolvent");
    addProfile.addItem("Add epicyloid");
    addProfile.addItem("Add polynomial curve");
}

JButton editSegment              = new JButton("Edit selected segment");
JButton inspectSegment           = new JButton("Inspect selected segment");
JButton refresh                  = new JButton("Refresh");
JButton calculateErrors          = new JButton("<html>Calc.<br> err.");
JButton displayTable             = new JButton("<html>Display<br>table<br>values");

JButton load                     = new JButton("Load");
JButton save                     = new JButton("Save");


JToggleButton zoomIn             = new JToggleButton("(+)");
JToggleButton zoomOut            = new JToggleButton("(-)");

JSlider[] lambda                 = null;

public JTextArea infos           = new JTextArea();
public JList segments            = new JList();


/**
 * Reference to controlled graphic canvas.
 */
protected GraphicCanvas graphicCanvas;

public Dimension getPreferredSize() {
    return new Dimension(500, 200);
}

public Toolbar() {
    //custom layout
    setLayout(null);
    setBackground(Color.WHITE);

    //add zoom
    zoomLabel.setBounds(0, 0, 100, 20);
    this.add(zoomLabel);
    zoomSpinner.setBounds(0, 20, 100, 20);
    this.add(zoomSpinner);
    zoomSpinner.addChangeListener(this);

    //add polyDegree
    radiusLabel.setBounds(0, 40, 100, 20);
    this.add(radiusLabel);
    radiusSpinner.setBounds(0, 60, 100, 20);
    this.add(radiusSpinner);
    radiusSpinner.addChangeListener(this);

    //add polyDegree
    ployDegreeLabel.setBounds(0, 90, 100, 20);
    this.add(ployDegreeLabel);
    polyDegreeSpinner.setBounds(0, 110, 100, 20);
    this.add(polyDegreeSpinner);
    polyDegreeSpinner.addChangeListener(this);

    //show moves
    showMoves.setBounds(110, 0, 180, 20);
    this.add(showMoves);
    showMoves.addActionListener(this);

    addProfile.setBounds(110, 20, 180, 20);
    this.add(addProfile);
    addProfile.addActionListener(this);

    editSegment.setBounds(110, 100, 180, 20);
    this.add(editSegment);
    editSegment.addActionListener(this);

    inspectSegment.setBounds(110, 60, 180, 20);
    this.add(inspectSegment);
    inspectSegment.addActionListener(this);

    //add clearProfileButton
```

```
            clearProfileButton.setBounds(110,120,180,20);
            clearProfileButton.addActionListener(this);
            this.add(clearProfileButton);

            //add refresh
            refresh.setBounds(110,140,180,20);
            refresh.addActionListener(this);
            this.add(refresh);

            zoomIn.setBounds(0,130,50,20);
            zoomIn.addChangeListener(this);
            this.add(zoomIn);

            zoomOut.setBounds(50,130,50,20);
            zoomOut.addChangeListener(this);
            this.add(zoomOut);


            initLambdas();


            JScrollPane scrollPaneSegments = new JScrollPane(segments);
            scrollPaneSegments.setBounds(300, 0, 300, 150);
            this.add(scrollPaneSegments);

            JScrollPane scrollPaneInfos = new JScrollPane(infos);
            scrollPaneInfos.setBounds(600, 0, 300, 150);
            this.add(scrollPaneInfos);

            calculateErrors.setBounds(900, 0, 100, 40);
            this.add(calculateErrors);
            calculateErrors.addActionListener(this);

            displayTable.setBounds(900, 40, 100, 60);
            this.add(displayTable);
            displayTable.addActionListener(this);

            load.setBounds(900, 100, 100, 20);
            this.add(load);
            load.addActionListener(this);

            save.setBounds(900, 120, 100, 20);
            this.add(save);
            save.addActionListener(this);
        }

        java.util.List theoreticalProfile = new java.util.ArrayList();

        /**
         * Getter for field: theoreticalProfile.
         * @return value of field: theoreticalProfile.
         */
        public java.util.List getTheoreticalProfile() {
            return theoreticalProfile;
        }

        /**
         * Find out zoom value contained by zoom spinner component.
         * @return zoom value
         */
        public double getZoomValue() {
            return ((Double) zoomSpinner.getValue()).doubleValue() / 100;
        }

        public void setGraphicCanvas(GraphicCanvas graphicCanvas) {
            this.graphicCanvas = graphicCanvas;
        }

        double oldZoomValue = 100;
        public void stateChanged(ChangeEvent e) {
            if (e.getSource() == zoomSpinner) {
                graphicCanvas.isModifying = true;
                graphicCanvas.scrollcanvasContainer.getVerticalScrollBar().setValue(
                        (int)
((graphicCanvas.scrollcanvasContainer.getVerticalScrollBar().getValue()       +       200)       *
getZoomValue()/oldZoomValue ) - 200
                );
```

```
                graphicCanvas.scrollcanvasContainer.getHorizontalScrollBar().setValue(
                        (int)
((graphicCanvas.scrollcanvasContainer.getHorizontalScrollBar().getValue()       +       500)       *
getZoomValue()/oldZoomValue ) - 500
                );
                graphicCanvas.isModifying = false;

                graphicCanvas.repaint();
                graphicCanvas.validate();
                zoomSpinner.validate();
                oldZoomValue = getZoomValue();
            }
            if (e.getSource() == radiusSpinner) {
                for (int i = 0; i < theoreticalProfile.size(); i++) {
                    ((TheoreticalProfile) theoreticalProfile.get(i)).resize(getRadiusRatio());
                }
                radiusSpinner.repaint();
                radiusSpinner.validate();
                radiusSpinner.validate();
                segments.validate();
                segments.repaint();
                graphicCanvas.repaint();
                graphicCanvas.validate();
            }
            if (e.getSource() == zoomSpinner) {
                graphicCanvas.repaint();
                graphicCanvas.validate();
                zoomSpinner.validate();
            } else if (e.getSource() == polyDegreeSpinner) {
                initLambdas();
                graphicCanvas.repaint();
                graphicCanvas.validate();
                polyDegreeSpinner.validate();
                this.repaint();
                this.validate();
            } else {
                //lambda changed;
                graphicCanvas.repaint();
                graphicCanvas.validate();
            }
        }

        private void initLambdas() {
            int degree = ((Double) polyDegreeSpinner.getValue()).intValue();
            if (lambda != null) {
                for (int i = 0; i < lambda.length; i++) {
                    this.remove(lambda[i]);
                }
            }

            lambda = new JSlider[degree + 1];
            lambda[0] = new JSlider(0, 1000, 0);
            lambda[degree] = new JSlider(0, 1000, 1000);
            for (int i = 1; i < degree; i++) {
                lambda[i] = new JSlider(0, 1000, i * 1000 / degree);
            }

            for (int i = 0; i <= degree; i++) {
                lambda[i].setBounds(500, i*20, 300, 20);
                //this.add(lambda[i]);
                lambda[i].addChangeListener(this);
            }
        }

        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == clearProfileButton) {
                this.theoreticalProfile = new ArrayList();
                this.segments.setListData(this.theoreticalProfile.toArray());
                this.graphicCanvas.resetCircleParam();
            } else if (e.getSource() == addProfile && addProfile.getSelectedIndex() == 1) {
                addProfile.setSelectedIndex(0);
                new DialogSegment(this);
                this.repaint();
                this.validate();
            } else if (e.getSource() == addProfile && addProfile.getSelectedIndex() == 2) {
                addProfile.setSelectedIndex(0);
                new DialogArc(this);
```

```
                this.repaint();
                this.validate();
            } else if (e.getSource() == addProfile && addProfile.getSelectedIndex() == 3) {
                addProfile.setSelectedIndex(0);
                new DialogEvolvent(this);
                this.repaint();
                this.validate();
            } else if (e.getSource() == addProfile && addProfile.getSelectedIndex() == 4) {
                addProfile.setSelectedIndex(0);
                new DialogEpicic(this);
                this.repaint();
                this.validate();
            } else if (e.getSource() == addProfile && addProfile.getSelectedIndex() == 5) {
                addProfile.setSelectedIndex(0);
                new DialogPoly(this);
                this.repaint();
                this.validate();
            } else if (e.getSource() == displayTable) {
                DisplayTableValuesDialog          displayTableValuesDialog          =          new
DisplayTableValuesDialog(this);
                displayTableValuesDialog.setModal(true);
                displayTableValuesDialog.setVisible(true);
            } else if (e.getSource() == calculateErrors) {
                this.infos.setText("");
                PolynomialCurve.report = "";
                for (int tp = 0; tp < this.getTheoreticalProfile().size(); tp++) {
                    TheoreticalProfile       theoreticalProfile       =       (TheoreticalProfile)
this.getTheoreticalProfile().get(tp);

                    PolynomialCurve polCurve = null;
                    if(theoreticalProfile.getProfilePointsList().size() > 1) {
                        polCurve = new PolynomialCurve(
                                ((Double) this.polyDegreeSpinner.getValue()).intValue(), this
                        );

                        polCurve.interpolate(
                                theoreticalProfile,
                                this.getLambdaValues()
                        );
                        polCurve.calculate(50);
                    }
                }

                try {
                    FileOutputStream fos = new FileOutputStream(
                            System.getProperty("user.home") +
                            "\\" +
                            "report.html"
                    );
                    fos.write(PolynomialCurve.report.getBytes());
                    fos.flush();
                    fos.close();
                } catch (Exception e1) {
                    e1.printStackTrace();
                }

                this.repaint();
                this.validate();
            } else if (e.getSource() == save) {
                JFileChooser fc = new JFileChooser();
                if (fc.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
                    try {
                        FileOutputStream fos = new FileOutputStream(fc.getSelectedFile());
                        ObjectOutputStream oos = new ObjectOutputStream(fos);
                        oos.writeObject(theoreticalProfile);
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                }
            } else if (e.getSource() == load) {
                JFileChooser fc = new JFileChooser();
                if (fc.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
                    try {
                        FileInputStream fos = new FileInputStream(fc.getSelectedFile());
                        ObjectInputStream oos = new ObjectInputStream(fos);
                        theoreticalProfile = (java.util.List) oos.readObject();
                    } catch (Exception e1) {
```

```
                    e1.printStackTrace();
                }
                this.segments.setListData(theoreticalProfile.toArray());
                this.graphicCanvas.repaint();
            }
        } else if (e.getSource() == editSegment) {
            int index = segments.getSelectedIndex();
            if (index >= 0) {
                if (((TheoreticalProfile) theoreticalProfile.get(index)).radious == -1) {
                    //edit segment
                    DialogSegment dialogSegment = new DialogSegment(this, index);
                } else {
                    //edit arc
                    DialogArc dialogSegment = new DialogArc(this, index);
                }
                this.repaint();
                this.validate();
            }
        } else if (e.getSource() == inspectSegment) {
            int index = segments.getSelectedIndex();
            if (index >= 0) {
                DialogInspection dialogSegment = new DialogInspection(this, index);
                this.repaint();
                this.validate();
            }
        }
        this.graphicCanvas.repaint();
    }

    public double[] getLambdaValues() {
        int degree = ((Double) polyDegreeSpinner.getValue()).intValue();
        double[] retValue = new double[degree + 1];
        for (int i = 0; i <= degree; i++) {
            retValue[i] = ((double) lambda[i].getValue()) / 1000;
        }
        return retValue;
    }

    public void setLambdaMiddle(double lambdaValue) {
        lambda[1].setValue((int) (lambdaValue * 1000));
    }


    public double getRadiusRatio() {
        return (((Double) radiusSpinner.getValue())) / TheoreticalProfile.PROFILE_RADIOUS;
    }

    public double getRadius() {
        return (((Double) radiusSpinner.getValue()));
    }
}
```

```
package ro.ugal.profiles.approximation.ui;

import javax.swing.*;
import javax.swing.event.ChangeListener;
import java.util.ArrayList;
import java.util.List;

public class ZoomSpinnerModel implements SpinnerModel {

    private static final double STEP = 1.0d;

    double zoomValue = 100;
    private List listeners = new ArrayList();

    public Object getNextValue() {
        zoomValue += STEP;
        return new Double(zoomValue);
    }

    public Object getPreviousValue() {
        zoomValue -= STEP;
        return zoomValue > 0 ? new Double(--zoomValue) : new Double(0);
    }
```

```
    public Object getValue() {
        return new Double(zoomValue);
    }

    public void setValue(Object value) {
        if (value instanceof Double) {
            zoomValue = ((Double) value).doubleValue();
        } else {
            zoomValue = 100;
        }
    }

    public void addChangeListener(ChangeListener l) {
        listeners.add(l);
    }

    public void removeChangeListener(ChangeListener l) {
        listeners.remove(l);
    }
}
```